sinclair

ZX81

PERSONAL COMPUTER

DRION

COMPUTERS

sinclair

ZX81
PERSONAL COMPUTER

DRION
COMPUTERS

Table des matières

L'INSTALLATION DU ZX81

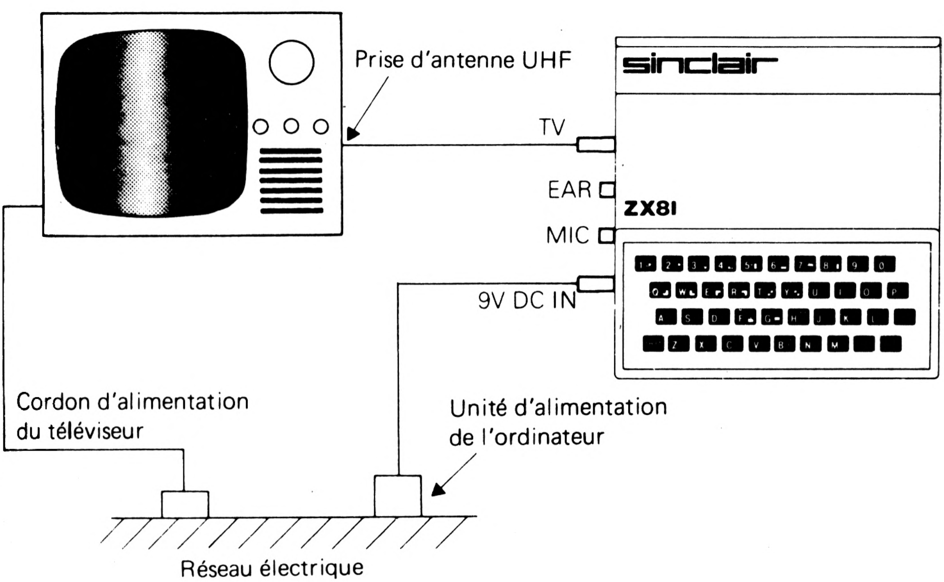
En déballant le ZX81, vous avez trouvé:

1. Ce manuel;

2. L'ordinateur. Celui-ci a trois prises (marquées 9 V DC IN, EAR, MIC), une prise d'antenne et la partie exposée du panneau de circuits, auquel vous pouvez ajouter un équipement supplémentaire. Il n'y a pas d'interrupteur - pour la mise en marche de l'ordinateur, il suffit de le brancher.

3. Une alimentation électrique. Celle-ci transforme l'électricité courante en celle qui est utilisée par le ZX81. Si, par accident, vous la mettez dans la mauvaise prise de l'ordinateur, il n'y a aucun mal. Si vous voulez utiliser votre propre source d'électricité, vous devez accoupler l'ordinateur à 9 V DC de 700 mA avec une fiche positive de 3,5 mm.

4. Un câble d'antenne de 120 cm de long, qui connecte l'ordinateur à votre téléviseur.



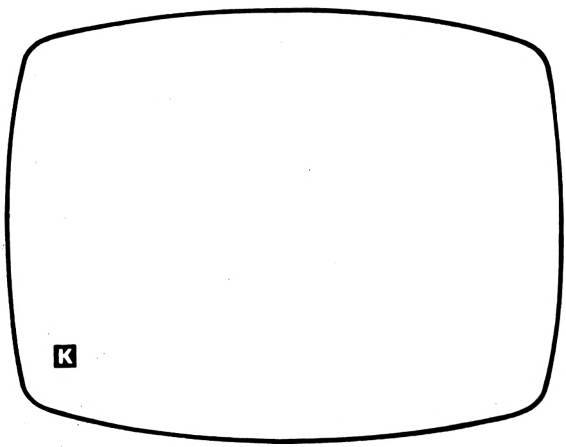
5. Une paire de câbles de 30 cm avec des fiches jack de 3,5 mm aux deux bouts.
Ceux-ci connectent l'ordinateur à l'enregistreur à cassettes. Vous aurez également besoin d'un téléviseur. L'ordinateur peut également fonctionner sans téléviseur, mais vous serez incapable de suivre ce qu'il fait. Il vous faut un téléviseur UHF. S'il ne capte pas UHF 36, il ne fait pas l'affaire.

Plus tard, il vous faudra également un enregistreur à cassettes, parce que, si vous éteignez le ZX81, toute l'information qu'il avait emmagasinée est perdue à tout jamais. Le seul moyen de garder cette information, est de l'enregistrer sur cassettes. Ceci est expliqué au chapitre 16. Vous pouvez également acheter des cassettes préparées par d'autres et exécuter ces programmes.

Si vous avez réuni toutes les parties de l'ordinateur, excepté l'enregistreur, vous les connectez, comme vous le montre le dessin de la page 0.

Si votre télévision a deux prises d'antenne (UHF et VHF), employez le UHF.

Branchez et allumez la télévision. Maintenant il faut encore la régler. Le ZX81 emploie le canal 36 de UHF. Quand il est branché et réglé, vous recevez cette image:



Quand vous utilisez l'ordinateur, vous voudrez probablement baisser directement le son du téléviseur. S'il est pourvu d'un contrôle permanent variable de réglage, vous devez l'ajuster jusqu'à ce que vous recevez cette image. Presque toutes les télévisions ont des boutons de présélection; choisissez-en un qui n'est pas encore employé et réglez-le.

Si vous bloquez l'ordinateur, souvenez-vous que vous pouvez toujours le débloquer et retourner vers cette image en retirant la fiche 9 V DC IN et en la remettant par la suite. Ceci est un moyen ultime, car vous perdez l'information emmagasinée dans l'ordinateur.

Note: cette description de la télévision s'applique seulement au Royaume Uni, où il existe un système UHF qui emploie 625 lignes à 50 images par seconde. C'est presque identique dans tous les autres pays (par exemple presque tous les pays occidentaux excepté la France). Les Etats-Unis emploient le VHF et 525 lignes à 60 images par seconde.

Maintenant que vous avez installé l'ordinateur, vous voulez sans doute l'utiliser. Si vous connaissez déjà le langage d'ordinateur BASIC, vous pouvez directement passer à l'annexe C et employer le reste de ce manuel pour éclaircir les points obscurs.

Si vous êtes un novice, alors la partie la plus importante a été écrite pour vous. N'ignorez pas les exercices; vous y trouverez des points importants dont on ne fait pas allusion dans le texte. Regardez-les et faites ceux qui vous intéressent ou qui se réfèrent à des sujets que vous ne comprenez pas très bien.

Surtout, employez l'ordinateur. Si vous avez une question, par ex. "qu'est-ce qui se passe quand je fais ceci ou cela?", la réponse est simple: écrivez-la et lisez la réponse. Chaque fois que le manuel vous ordonne de taper quelque chose, vous devez vous demander "que pourrai-je faire à la place?" et exécuter les réponses.

Plus vous écrirez vos propre réflexions, mieux vous comprendrez le ZX81 (ceci s'appelle un apprentissage non-programmé). Ce que vous écrivez n'a aucune importance, vous ne pouvez pas endommager l'ordinateur.

DONNEZ VOS INSTRUCTIONS A L'ORDINATEUR

Allumez l'ordinateur (mettez la fiche) et vous avez l'image blanche avec le K blanc sur fond noir, comme sur le dessin du chapitre I. Afin de faire fonctionner l'ordinateur, vous devez lui donner une instruction qu'il comprenne.
Par exemple l'instruction:

PRINT 2 + 2

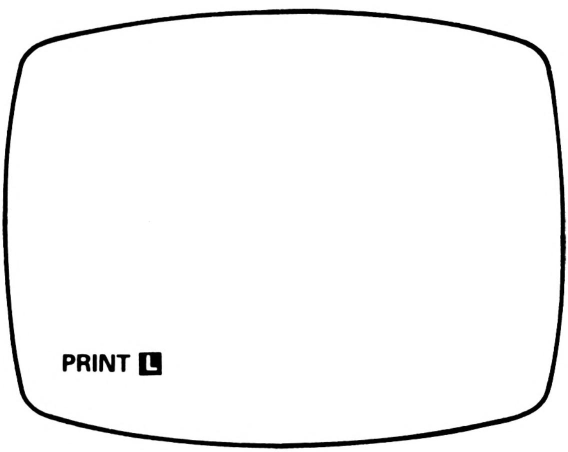
ordonne de calculer la somme 2 + 2 et de visualiser la réponse sur l'écran de la télévision. Un message comme celui-là, qui dit à l'ordinateur de faire quelque chose tout de suite, est une instruction.

Celle-ci est une instruction **PRINT** (imprimer), mais elle est en même temps une déclaration **PRINT**. On l'appelle une déclaration **PRINT** parce qu'elle spécifie la forme sans dire comment l'ordinateur doit employer cette déclaration.

Donc chaque instruction a la forme d'une déclaration. Mais ceci vaut également pour d'autres choses, comme pour par exemple les lignes de programme (cfr. chapitre 8).

Afin d'introduire cette instruction:

1. Enfoncez d'abord **PRINT**. Mais, même si vous voyez une touche pour chaque lettre vous ne devez pas écrire le mot entier P,R,I,N,T,. En enfonçant simplement la lettre P, le mot paraîtra entièrement sur l'écran avec des espaces, pour rendre le tout plus ordonné. L'écran montrera:



La raison est que l'ordinateur s'attend à recevoir un mot clef au début de chaque instruction; un mot qui explique de quelle commande il s'agit. Les mots clefs sont écrits au-dessus des touches et vous verrez que **PRINT** est marqué au-dessus de la touche P. Ainsi, vous n'avez qu'à enfoncer la touche P pour que le mot **PRINT** apparaîsse.

L'ordinateur vous fait savoir qu'il attend le mot clef en vous montrant le [K], la lettre avec laquelle vous avez commencé. Vous trouverez presque toujours une lettre blanche sur fond noir, un [K] ou un [L] (ou comme on verra plus tard, un [F] ou un [G]), qui s'appelle curseur. Le [K] veut dire: "n'importe ce que vous écrivez, je l'interprète comme un mot clef". Comme vous avez déjà vu, le [K] fait place à un [L] après avoir enfoncé P pour **PRINT**.

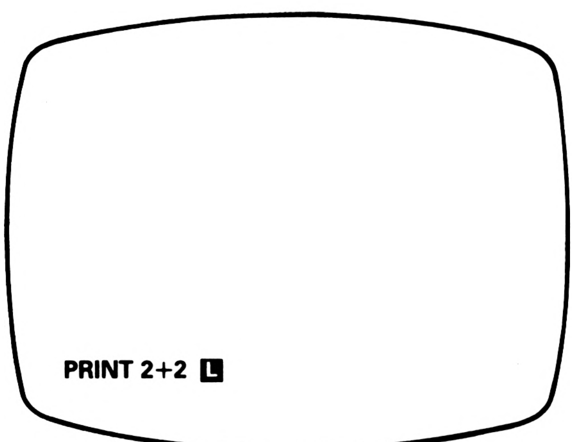
Ce système d'enfoncer une seule touche pour avoir plus qu'un symbole, est fréquemment employé sur le ZX81. Dans la suite de ce manuel, les mots avec leur propre clef seront imprimés en **CARACTERES GRAS.**

Souvenez-vous qu'il est inutile d'épeler ces mots lettre par lettre, car l'ordinateur ne les comprendra tout de même pas.

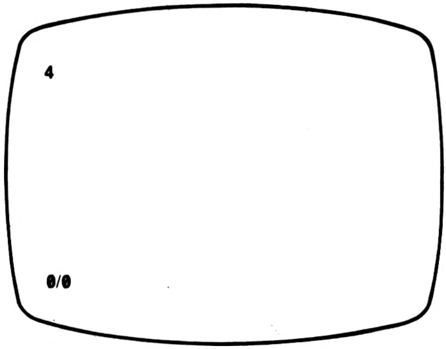
2. Ecrivez maintenant 2. Cela ne posera aucun problème. Vous verrez, de nouveau, apparaître le 2 sur l'écran et la lettre L reculera d'une place. Remarquez également qu'il y a un espace entre **PRINT** et le chiffre 2. Cet espace donne à l'ensemble une impression ordonnée. Ceci se fait aussi souvent que possible pour que vous n'ayez pas à laisser un espace. Si vous écrivez un espace, il apparaîtra sur l'écran, mais il n'affectera aucunement le message.

3. Ecrivez maintenant le +. Ceci est un caractère décalé (shifted character) - ils sont marqués en rouge, la couleur de la touche SHIFT, en haut à droite de chaque touche. Pour avoir le "+", il faut enfoncer la touche SHIFT en même temps que la touche K.

4. Ecrivez encore une fois 2. L'écran sera comme ceci:



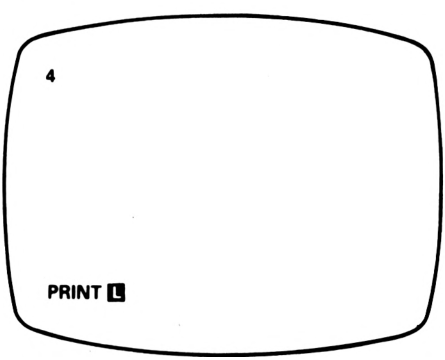
5. Maintenant, souvenez-vous-en, enfoncez la touche NEWLINE.
Elle veut dire: “message terminé” ou “en marche, l'ordinateur, faites faire quelque chose”. L'ordinateur lit le message, calcule ce qu'il doit faire et exécute les instructions. En l'occurence, l'écran sera comme ci-dessous:



La réponse est 4. Evidemment, vous n'avez pas besoin d'un ordinateur pour trouver ce résultat.
ØØ (remarquez que le zéro est barré pour le différencier de la majuscule O, ce qui est commun pour les ordinateurs) est le rapport dans lequel l'ordinateur vous raconte comment il a fonctionné.

Le premier Ø signifie: “OK, pas de problèmes” (dans l'annexe B il y a une liste avec les autres codes de rapports que vous pouvez rencontrer, par ex. quand quelque chose va mal). Le second Ø signifie: “la dernière chose que j'ai faite est la ligne Ø”.

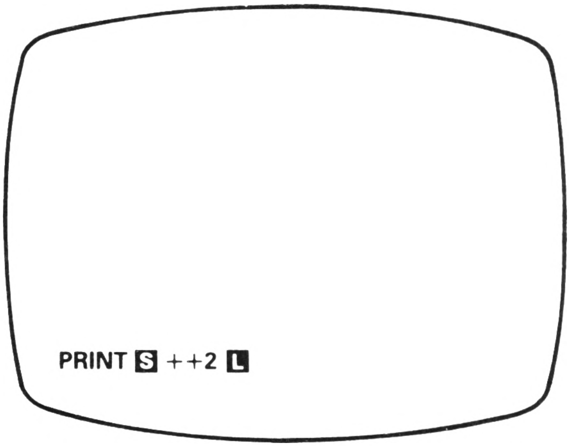
Plus tard, quand vous allez écrire des programmes, vous verrez qu'une déclaration peut recevoir un numéro. Ainsi, elle peut être rangée et réemployée plus tard. On parle alors de ligne de programme. Les instructions n'ont pas encore de numéros, mais l'ordinateur leur donne le chiffre Ø afin de pouvoir les rapporter. Imaginez que le rapport cache le curseur K. Si vous enfoncez la touche P pour **PRINT**, ce rapport disparaîtra et l'écran sera comme ci-dessous.



Le curseur peut également être employé pour corriger des fautes.
Ecrivez ++2 et l'écran montre:

PRINT ++2 **L**

sur la ligne de fond. C'est un message plutôt incompréhensible, mais quand vous enfoncez **NEWLINE**, vous avez:



Le **S** est le marqueur des erreurs de syntaxe (la syntaxe est la grammaire des messages: elle dit quels messages sont admis et lesquels ne le sont pas) et montre que l'ordinateur a bien reçu le **PRINT** mais qu'il a décidé que ce message n'est pas correct.

Vous voulez sans doute effacer le premier + et le remplacer par 3 par exemple. D'abord vous devez déplacer le curseur afin qu'il soit directement à la droite du premier +. Il y a deux touches ⇦ et ⇨ (shift + 5 et shift + 8) qui déplacent le curseur vers la gauche et vers la droite. Enfoncez la touche **SHIFT** et poussez deux fois sur la touche ⇦. Le curseur recule de deux places:

PRINT + **L** + 2

Enfoncez maintenant la touche **RUBOUT** (**SHIFT** + Ø) et vous obtenez:

PRINT **L** + 2

RUBOUT efface les caractères ou les mots clefs situés à gauche du curseur. Si vous frappez le 3, vous insérez un "3", également à gauche du curseur. Ceci donne:

PRINT 3 **L** + 2

Enfoncez **NEWLINE** et vous recevez la réponse 5.

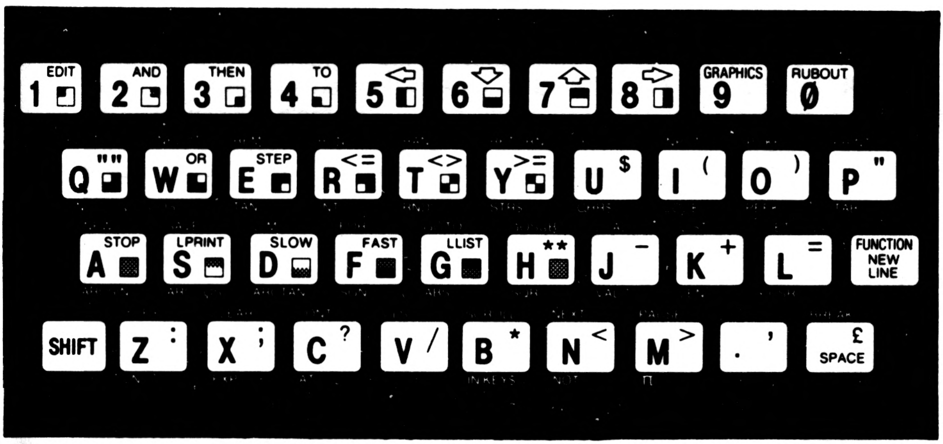
La touche ⇨ (**SHIFT** + 8) fonctionne comme la touche ⇦ , excepté qu'il déplace le curseur vers la droite au lieu de vers la gauche.

Sommaire

Ce chapitre vous a montré comment introduire des messages dans le ZX81 en expliquant le fonctionnement du système qui consiste à frapper une seule touche afin de recevoir un mot entier;
les curseurs **K** et **L**;
les rapports;
le marqueur d'erreur de syntaxe **S**
et comment corriger les erreurs en employant ⇦ , ⇨ et **RUBOUT**.

Le clavier

Voici une photo du clavier:



Souvenez-vous que pour l'emploi de **SHIFT**, il faut enfoncer cette touche en même temps qu'une autre touche. Ne confondez pas Ø avec la lettre O.

UNE LEÇON D'HISTOIRE

Les messages que vous imprimez sont écrits en langue d'ordinateur **BASIC** (Beginners' All Purpose Symbolic Instruction Code, c.-à-d. le langage des débutants).

L'ordinateur doit fournir un effort considérable pour traduire les messages en **BASIC** en ses propres opérations rudimentaires. Mais, après tout, c'est pour cela qu'on le paie. Les messages en **BASIC** contiennent assez de mots anglais (par exemple **PRINT**) pour être compris par quelqu'un qui connaît l'anglais.

Le **BASIC** a été inventé au collège de Dartmouthau New Hampshire, USA, en 1964, et depuis lors, il est le langage le plus utilisé par les débutants et les profanes. Surtout parce qu'il s'adapte très bien à l'emploi "on-line", c.-à-d. que quand l'usager donne un message, l'ordinateur répond immédiatement.

Il existe encore d'autres langages, par exemple **ALGOL**, **PASCAL**, avec une structure beaucoup mieux établie et une plus grande capacité que le **BASIC**, mais il n'y a pas beaucoup de ces autres langages qui peuvent utiliser le système "on-line".

Les autres langages à ne pas oublier sont: le **COBOL**, le **FORTRAN** et le **PL1**. Beaucoup de magazines d'ordinateurs publient des programmes en Basic. Ils valent la peine d'être lus. Vous aurez probablement à les adapter un peu car chaque ordinateur qui emploie le **BASIC**, a son propre dialecte qui se différencie des autres.

LE SINCLAIR ZX81 COMME MACHINE A CALCULER

Branchez l'ordinateur. Maintenant vous pouvez l'employer comme machine à calculer, comme il est expliqué au chapitre 2: frappez **PRINT**, introduisez ensuite ce que vous voulez calculer, puis frappez **NEWLINE** (en général nous ne mentionnerons plus le **NEWLINE**).

Comme vous l'avez espéré, le ZX81 n'est pas seulement capable d'additionner, mais également de multiplier, de soustraire et de diviser (multiplier = * et diviser = /)

+, -, * et / sont des opérations et les chiffres sur lesquels ils opèrent sont les opérandes.

L'ordinateur est capable de calculer des puissances à l'aide de l'opération ** (= **SHIFT** + H, n'employez pas la touche * - **SHIFT** + B - 2 fois); écrivez maintenant:

PRINT 2**3 (n'oubliez pas **NEWLINE**)

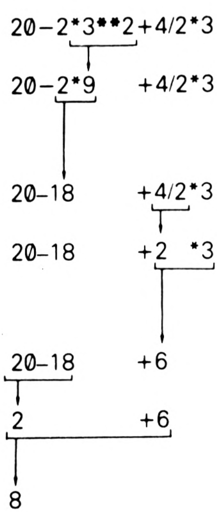
et vous obtenez le résultat 8 (2^3).

Le ZX81 sait également calculer des opérations combinées. Par exemple:

PRINT 2Ø - 2*3**2 + 4/2*3

donne le résultat 8. Il fait tout un détour pour obtenir ce résultat; d'abord, il calcule toutes les puissances (**) dans l'ordre de gauche à droite, ensuite les multiplications et les divisions (* et /) aussi de gauche à droite et enfin les additions et les soustractions (+ et -) également de gauche à droite.

Ainsi notre exemple est calculé en étapes:



d'abord les puissances,

ensuite les multiplications et les divisions,

enfin les additions et les soustractions.

Nous concrétisons ce système en donnant une priorité à chaque opération, c.-à-d. un chiffre entre 1 et 16. L'opération avec la plus grande priorité est évaluée en premier lieu, et les opérations avec les mêmes priorités sont évaluées de gauche à droite.

**	a une priorité de 1Ø.
* et /	ont une priorité de 8.
+ et -	ont une priorité de 6.

Au moment ou "-" est employé comme signe de négation, par exemple -1, il a une priorité de 9. C'est un signe de négation "unaire" contraire au signe de négation "binaire" dans 3 - 1 : une opération unaire n'a qu'un seul opérande, là où une opération binaire en a deux. Sachez qu'avec le ZX81, il est impossible d'employer l'opération + comme opération unaire.

Cet ordre est strict et ne peut être changé qu'en employant des parenthèses.

Ce qui se trouve entre parenthèses est évalué en premier lieu et considéré comme chiffre entier. Ainsi:

PRINT 3*2 + 2

donne 6 + 2 = 8. Mais:

PRINT 3*(2 + 2)

donne 3*4 = 12

Une telle combinaison s'appelle une expression - dans ce cas il s'agit d'une expression arithmétique ou numérique, puisque le résultat est un chiffre.

En général, quand l'ordinateur s'attend à un chiffre, vous pouvez lui donner une expression au lieu d'un chiffre et il vous donnera le résultat.

Vous pouvez écrire des chiffres décimaux (utilisez le point), et vous pouvez également utiliser des symboles scientifiques - comme sur les machines à calculer de poche.

Ainsi, après un chiffre normal (avec ou sans point décimal), vous pouvez écrire une partie exponentielle qui consiste en une lettre E suivie d'un chiffre, avec ou sans point décimal, positif ou négatif. La lettre E exprime "*1Ø**" (multiplié par $1Ø^n$). Ainsi:

2.34EØ	= 2.34*1Ø**Ø	=2.34
2.34E3	= 2.34*1Ø**3	=234Ø
2.34E-2	= 2.34*1Ø**-2	= Ø.Ø234

(Essayez d'exécuter cet exemple sur le ZX81).

Afin de comprendre cette façon de calculer, il faut s'imaginer que la partie exponentielle fait avancer le point décimal vers la droite (s'il s'agit d'un exponent positif) ou reculer vers la gauche (s'il s'agit d'un exponent négatif).

Il est également possible d'écrire plus qu'une commande à la fois en les séparant par une virgule ou un point-virgule (**SHIFT** + X).

En employant la virgule, le chiffre suivant se montre à la marge de gauche ou au milieu de la ligne à la 16ième colonne.

En employant le point-virgule, le chiffre suivant suit immédiatement le chiffre précédent. Essayez:

PRINT 1; 2; 3; 4; 5; 6; 7; 8; 9; 10

&

PRINT 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

afin de voir la différence. Il est même possible de mélanger des points et des points-virgules dans une seule déclaration.

Sommaire

Déclarations: **PRINT**, avec virgules et points-virgules.
Opérations: +, −, *, /, **.
Expressions, symboles scientifiques.

Exercices

1. Essayez:

PRINT 2.34EØ
PRINT 2.34E1
PRINT 2.34E2

PRINT 2.34E15

Vous verrez qu'après quelque temps, le ZX81 commence à employer des symboles scientifiques puisqu'il n'est pas capable d'écrire un chiffre de plus de 14 espaces. Essayez de la même façon:

PRINT 2.34E-1
PRINT 2.34E-2

etc...

2. Essayez:

PRINT 1,,2,,3,,,4,,,,5

Une virgule déplace toujours le curseur juste avant le chiffre suivant. Essayez maintenant:

PRINT 1;;2;;3;;;4;;;;5

Pourquoi une chaîne de points-virgules ne diffère-t-elle pas d'un seul?

3. **PRINT** ne donne que 8 caractères significatifs. Essayez:

PRINT 429467295, 4294967295-429E7

Cela prouve que l'ordinateur peut mémoriser tous les caractères de 4294967295, même s'il n'est pas capable de les montrer tous à la fois.

4. Quand vous possédez des tables de logarithmes, il est possible de contrôler la règle suivante:

PRINT 1Ø**Ø,3Ø1Ø

et vérifiez dans vos tables.

5. Le ZX81 utilise l'arithmétique de la virgule flottante ce qui veut dire qu'il sépare les caractères des chiffres (mantisse) de la position du point (l'exponent). Cela n'est pas toujours exact, même pas pour les chiffres entiers.
Ecrivez:

PRINT 1E1Ø + 1 - 1E1Ø - 1E1Ø - 1E1Ø + 1

Le ZX81 peut montrer avec une exactitude totale des chiffres d'environ 9 1/2 caractères; ainsi, 1E1Ø est trop grand pour être montré avec précision.

L'imprécision est plus grande que 1 (en fait, elle est d'environ 2), ainsi les chiffres 1E1Ø et 1E1Ø + 1 semblent être égaux pour l'ordinateur.
Pour avoir un exemple encore plus spécial, écrivez:

PRINT 5E9 + 1 - 5E9

Dans cet exemple, l'imprécision de 5E9 est seulement d'environ 1 et le 1 additionné est en fait arrondi jusqu'à 2. Ici les chiffres 5E9 + 1 et 5E9 + 2 semblent être identiques pour l'ordinateur. Le plus grand chiffre entier qui peut être montré avec précision est $2^{32}-1$ (4.294.967.295).

FONCTIONS

Mathématiquement, une fonction est une règle qui donne un chiffre (le résultat) en échange d'un autre chiffre (l'argument ou l'opérande). C'est donc une opération unaire.

Le ZX81 en possède quelques unes et leurs noms sont les mots écrits en-dessous des touches. **SQR** est la fonction de la racine carrée. Ainsi

PRINT SQR 9

donne le résultat 3, la racine carrée de 9. (pour avoir **SQR**, il faut enfoncer la touche **FONCTION** - **SHIFT** + **NEWLINE**. Ceci change le curseur en ■. Enfoncez maintenant **SQR** (H). **SQR** apparaît sur l'écran et le curseur change en ■. La même méthode vaut pour tous les mots écrits en-dessous des touches. Ce sont presque tous des noms de fonctions). Essayez:

PRINT SQR 2

Vous pouvez contrôler l'exactitude de votre réponse en imprimant:

PRINT SQR 2* **SQR** 2

ce qui vous donne la réponse 2. Notez que les deux **SQR** sont calculés avant la * multiplication. En fait, toutes les fonctions sont calculées avant +, –, *, / et ** (excepté pour **NOT**). On peut également éviter cette règle en mettant des parenthèses.

PRINT SQR (2*2)

donne 2.

Voici quelques autres fonctions (il y a une liste complète dans l'annexe C). Si vos connaissances en math ne vous permettent pas de les comprendre, cela ne fait rien, vous serez toujours capable d'utiliser l'ordinateur.

SGN: la fonction des signes (appelée souvent signum afin de ne pas confondre avec **SIN**). Le résultat est –1, Ø ou +1 selon la valeur de l'argument: négatif, zéro ou positif.

ABS: la valeur absolue ou module. Le résultat est l'argument positif.

ABS — 3.2 = **ABS** 3.2 = 3.2

SIN
COS
TAN
ASN arcsin
ACS arccos
ATN arctan

} Les fonctions trigonométriques. Ceux-ci fonctionnent en radians et pas en degrés.

LN Le logarithme naturel (base 2.718281828459Ø45..., alias e)
EXP Fonction exponentielle.

SQR La racine carrée.
INT La partie entière. Cette fonction arrondit toujours. **INT** 3.9 = 3 et **INT** -3.9 = -4.
PI π = 3,14159265358979. **PI** n'a pas d'argument. (Dix chiffres seulement sont emmagasinés dans l'ordinateur et huit seulement en sont montrés).
RND **RND** n'a pas d'argument non plus. Cette fonction vous donne un chiffre arbitraire entre Ø (un chiffre que cette fonction accepte) et 1 (qui n'est pas accepté).

Quand on emploie le jargon du chapitre précédent, toutes ces fonctions, excepté la fonction **PI** et **RND**, sont des opérations unaires avec une priorité de 11. (**PI** et **RND** sont des opérations nullaires parce qu'elles n'ont pas d'opérande).

Les fonctions de trigonométrie **EXP**, **LN** et **SQR** sont calculées à 8 chiffres près.
RND et **RAND**: se trouvent sur la même touche, mais **RND** est une fonction tandis que **RAND** est un mot clef, comme **PRINT**. **RAND** est utilisé pour contrôler si **RND** est arbitraire. **RND** n'est pas vraiment arbitraire, mais il suit une séquence fixe de 65536 chiffres qui peuvent paraître rassemblés de telle façon qu'on les croirait arbitraires (**RND** est pseudo arbitraire).

Vous pouvez employer **RAND** pour commencer la fonction **RND** à une place bien déterminée dans la séquence en écrivant **RAND** suivie d'un chiffre entre 1 et 65535. Enfoncez ensuite **NEWLINE**.

Il n'est pas tellement important de savoir où un chiffre donné introduit **RND**, puisque le même chiffre après **RAND** introduira toujours **RND** à la même place.
Ecrivez par ex.:

RAND 1 (& **NEWLINE**)

et ensuite:

PRINT RND

et écrivez-les plusieurs fois. (employez la touche **FONCTION** pour avoir **RND**).

La réponse de **RND** sera toujours Ø.ØØ22735596, ce qui n'est pas du tout une séquence arbitraire.

RAND Ø

(vous pouvez laisser tomber le Ø), se comporte un peu différemment. Il juge où commencer **RND** en se basant sur le temps que la télévision a été allumée; ceci est donc vraiment arbitraire.
Note: Dans certains dialectes du **BASIC**, il faut toujours enfermer l'argument d'une fonction entre parenthèses. Ceci n'est pas le cas pour le **BASIC** 8K du ZX81.

Sommaire

Déclaration: **RAND.**
Fonctions: **SNG, ABS, SIN, COS, TAN, ASN, ACS, ATN, LN, EXP, SQR, INT, PI, RND.**

FONCTION

Exercices:

1. Pour trouver le logarithme commun (base 1Ø), ce que vous pouvez rechercher dans vos tables de logarithmes, il faut diviser le logarithme naturel par **LN** 1Ø.
Pour trouver par exemple le log. de 2:

PRINT LN 2/**LN** 1Ø

ce qui vous donne: Ø.3Ø1Ø3.

Essayez de faire des multiplications et des divisions en employant les logarithmes, en employant le ZX81 comme table de logarithmes. Contrôlez vos réponses en employant * et /: c'est plus facile, plus vite et plus précis et donc à recommander.

2. **EXP** et **LN** sont des fonctions mutuellement inverses, en ce sens que si vous utilisez d'abord l'un et ensuite l'autre, vous recevez de nouveau votre chiffre de départ. Exemple:

LN EXP 2 = **EXP LN** 2 = 2

Ceci vaut également pour **SIN** et **ASN**, pour **COS** et **ACS**, pour **TAN** et **ATN**. Utilisez-les pour contrôler la précision de l'ordinateur.

3. Les radians de **PI** sont 180°. Donc pour convertir les degrés en radians, vous divisez par 180 et multipliez par **PI**. Ainsi:

PRINT TAN (45/180***PI**)

donne **TAN** 180° (1).

Pour passer de radians à degrés, vous divisez par **PI** et multipliez par 180.

4. Essayez

PRINT RND

quelque fois pour voir comment les résultats varient. Est-ce que vous découvrez une structure? (C'est peu probable).

Comment utiliseriez-vous **RND** et **INT** pour avoir un chiffre entier arbitraire entre 1 et 6 pour représenter le jet d'un dé? (Réponse: **INT** (**RND** 6) + 1).

5. Contrôlez cette règle:
Supposez que vous choisissez un chiffre entre 1 et 872 et écrivez:

RAND et ensuite votre chiffre (et **NEWLINE**)

La prochaine valeur de **RND** sera:

(75 * (votre chiffre + 1) –1) / 65536

6. Seulement pour mathématiciens.
Prenons p = un grand nombre premier et a = une racine primitive modulo p.
Quand b_i est le reste de a_i modulo p ($1 \leqslant b_i < p-1$), la séquence

$$\frac{b_i - 1}{p - 1}$$

est une séquence cyclique de p - 1 nombres différents dans la rangée de Ø à 1 (1 non compris). En choisissant bien la valeur de a, ceux-ci auront l'air bien arbitraires. 65537 est un facteur premier de Mersenne, $2^{16} - 1$.

En employant ce nombre et la loi de Gauss de réciprocité quadratique, vous pouvez calculer que 75 est une racine primitive modulo 65537.

Le ZX81 emploie p = 65537 et a = 75 et emmagasine des $b_i - 1$ dans sa mémoire.

La fonction **RND** implique le remplacement de $b_i - 1$ dans la mémoire par $b_{i+1} - 1$ et donne le résultat $(b_{i+1} - 1) / (p - 1)$. **RAND** n (avec $1 \leqslant n \leqslant 65535$) fait que $b_i = n + 1$.

7. **INT** arrondit toujours vers le bas. Pour arrondir vers le nombre le plus proche, ajoutez d'abord Ø.5. Par exemple:

INT (2.9 + Ø.5) = 3.	**INT** (2.9 + Ø.5) = 2
INT (-2.9 + Ø.5) = -3.	**INT** (-2.4 + Ø.5) = -2

Comparez ces réponses à celles que vous obtenez si vous n'ajoutez pas Ø.5.

8. Essayez:

PRINT PI, PI - 3, **PI** -3.1, **PI** - 3.14, **PI** - 3.141

Ceci démontre avec combien de précision l'ordinateur emmagasine **PI**.

VARIABLES

Vous objecterez que votre calculatrice de poche sait également emmagasiner des chiffres et s'en rappeler par la suite. Mais est-ce que le ZX81 sait le faire?

En effet, il peut en stocker des centaines en utilisant la déclaration **LET**.

Supposons que les œufs coûtent 58 Fr la douzaine et vous voulez vous en souvenir. Ecrivez:

LET EGGS = 58 (et **NEWLINE** évidemment)

D'abord, l'ordinateur cherchera un espace pour y emmagasiner un chiffre; ensuite, il donnera à cet espace le nom EGGS pour que vous puissiez vous y référer par après. Cette combinaison d'espace de stockage et de nom pour cet espace s'apelle une variable. Enfin, il a stocké le nombre 58 dans cet espace: on dit qu'il a assigné la valeur 58 à la variable (du nom de) EGGS. EGGS est une variable numérique parce que sa valeur est un chiffre.

Voulez-vous savoir combien coûtent les œufs? Tapez:

PRINT EGGS

Si vous désirez connaître la valeur d'une demi-douzaine d'œufs:

PRINT EGGS / 2

Si vous cherchez le carré du cosinus du prix d'un œuf:

PRINT COS (EGGS / 12) ** 2

Facile, n'est-ce pas? Mais que se passe-t-il quand votre personnel s'étonne: "Mon Dieu, le prix des œufs a passé à 61 Fr la douzaine". Il n'y a pas de temps à perdre. Ecrivez:

LET EGGS = 61

Ceci ne réserve pas un espace supplémentaire d'emmagasinage mais remplace l'ancienne valeur 58 par 61. Ecrivez maintenant:

PRINT EGGS

Ainsi, vous pouvez être certain de connaître le dernier prix.
Ecrivez maintenant:

PRINT MILK

Le rapport donnera: 2/Ø. En cherchant la valeur 2 dans l'index B, vous trouverez qu'elle signifie: "Variable non trouvée". L'ordinateur n'a pas la moindre idée du prix du lait, parce que vous ne lui avez pas dit. Ecrivez:

LET MILK = 18.5

et tout est en ordre.

Ecrivez:

PRINT MILK

Les variables n'ont pas nécessairement des noms d'épicerie: vous écrivez n'importe quoi pourvu que le nom commence par une lettre. Vous pouvez également laisser un espace pour rendre la lecture plus facile. Elle ne fera pas partie du nom.

Sont admis, par exemple, comme noms de variables:

TWO POUNDS OF APPLES BUT NOT GOLDEN DELICIOUS
(un kilo de pommes mais pas des Golden)
RADIO 3
RADIO 33
X
K9P

Mais pas:

3 BEARS (3 ours) parce que cela commence par un chiffre.
TALBOT? (? n'est pas une lettre ni un chiffre).
WHITE ON BLACK (les caractères en vidéo inversée ne sont pas admis)
FOTHERINGTON - THOMAS (- n'est pas une lettre ni un chiffre).

Ecrivez:

CLEAR

et

PRINT EGGS

Vous recevrez le rapport 2 (variable non trouvée).

L'effet de **CLEAR** est de relâcher tous les espaces qui ont été réservés aux variables. C'est comme si les variables n'avaient jamais été définies. Eteindre et rallumer l'ordinateur a le même effet. Il ne se souvient plus de rien au moment où vous voulez le réemployer.

Les expressions peuvent contenir un nom de variable partout où elles peuvent avoir un nombre.

Note: Dans certaines versions du BASIC, vous pouvez omettre **LET**; vous écrivez simplement:

EGGS = 58

Ceci n'est pas admis sur le ZX81. De toute façon, vous aurez des difficultés à l'introduire.

Dans certaines versions, seuls les deux premiers caractères d'un nom sont contrôlés. Alors, RADIO 3 est égal à RADIO 33. Dans d'autres versions, le nom de la variable doit être représenté par une lettre suivie d'un chiffre.

Aucune de ces restrictions ne s'applique au ZX81. A noter aussi que dans certaines versions du BASIC, si une variable n'est pas apparue à gauche de la déclaration **LET**, elle est supposée avoir la valeur Ø. Comme vous l'avez remarqué avec **PRINT** MILK, ceci n'est pas le cas avec le ZX81.

Sommaire:

VARIABLES.
DECLARATIONS: **LET** et **CLEAR**.

Exercices:

1. Pourquoi les noms des variables doivent-ils commencer par une lettre?

2. Si vous n'êtes pas accoutumés à calculer avec des puissances (** = shift + H), faites alors ces exercices.

A un niveau très élémentaire, A**B signifie A multiplié B fois par lui-même.

Evidemment, ceci n'a un sens que si B est un chiffre positif entier. Afin de trouver une définition qui s'applique à d'autres valeurs de B, nous considérons la formule suivante:

A ** (B + C) = A ** B * A ** C

Vous serez vite convaincus que ça marche si B et C sont des chiffres entiers et positifs, mais s'ils ne le sont pas, on est obligé d'accepter que:

A ** Ø = 1
A ** (—B) = 1 / A ** B
A ** (1/B) = la Bième racine de A
&
A ** (B*C) = (A ** B) ** C

Si vous n'avez jamais vu ça auparavant, retenez que:

A ** -1 = 1/A
A ** (1/2) = la racine carrée de A.

Dès que vous serez familiarisé avec ce qui précède, vous y verrez rapidement plus clair.

Expérimentez en demandant à l'ordinateur d'imprimer plusieurs expressions avec des **. Par exemple:

PRINT 3** (2 + Ø), 3**2*3**Ø
PRINT 4** -1, 1/4

3. Ecrivez:

LET E = **EXP** 1

E a maintenant la valeur de 2.718281828, la base des logarithmes naturels.

Expérimentez la règle:

EXP un chiffre = E** le chiffre

pour plusieurs chiffres.

LES CHAINES

Le ZX81 est capable de faire ce qu'une machine à calculer de poche ne sait pas faire. Ecrivez:

PRINT "HI THERE. I AM YOUR ZX81"

(" est SHIFT + P)

Ce bref salut placé entre guillemets s'appelle une chaîne (c'est-à-dire une chaîne de symboles) et peut contenir n'importe quel symbole, sauf le guillemet simple". (Mais vous pouvez employer ce qu'on appelle les symboles de citation, "" (SHIFT + Q) ce qui vous donne " lors d'une déclaration **PRINT**).

Une faute facilement commise, c'est qu'on oublie un des guillemets. Cette faute fait apparaître le marqueur d'erreur **S**. Quand vous imprimez des chiffres, vous pouvez employer ces chaînes afin d'expliquer le sens des chiffres.
Ecrivez par ex.:

LET EGGS = 61

Puis:

PRINT "the price of eggs is"; eggs; "new pence a dozen"

Ne faites pas attention à ce que la chaîne dépasse la ligne.

Cette déclaration vous montre trois choses: **PRINT** c.-à-d. la chaîne "the price of eggs is", le chiffre 61 (le prix de la variable, les œufs) et enfin la chaîne "new pence a dozen". En fait, vous pouvez imprimer n'importe combien d'éléments par chaîne et n'importe quel mélange de chaînes et de chiffres (ou expressions).

Remarquez comment les espaces entre les éléments d'une chaîne font aussi bien partie de la chaîne que les lettres. Elles ne sont pas oubliées, même pas à la fin d'une ligne.

Vous pouvez faire beaucoup de combinaisons avec des chaînes.

1. Vous pouvez les assigner à des variables. Néanmoins, la variable doit être spécialement marquée afin de montrer qu'elle a la valeur d'une chaîne et pas d'un chiffre: la lettre doit être suivie du symbole $ (SHIFT + U).
Ecrivez par ex.:

LET A$ = "Double Gloucester"

et

PRINT A$

2. Vous pouvez les combiner. Cela s'appelle enchaînement; l'ordinateur enchaîne vraiment les éléments. Essayez:

PRINT "GAMMO" + "N RASHERS"

Il est impossible de multiplier, de diviser ou de soustraire des chaînes ou les élever au carré.

3. Vous pouvez appliquer certaines fonctions aux chaînes afin d'obtenir des chiffres et vice versa.

- **LEN** peut être appliqué à une chaîne et résulte en une unité de longueur.
  Par ex.: **LEN** "Cheese" = 6
- **VAL** appliqué à une chaîne résulte en ce que donne la chaîne évaluée en expression arithmétique. Par exemple, (quand A = 9), **VAL** "1/2 + **SQR** A" = 3.5

Quand **VAL** est appliqué à une chaîne contenant des variables, il faut tenir compte de deux règles:

i) quand la fonction **VAL** fait partie d'une expression de plusieurs éléments, elle doit être à la première place.

10 **LET** X = 7 + **VAL** "Y" doit devenir 10 **LET** X = **VAL** "Y" + 7

ii) **VAL** ne peut apparaître qu'à la première coordonnée d'une déclaration **PRINT AT**, **PLOT** ou **UNPLOT** (cfr. chapitre 17 et 18).

Exemple: 10 **PLOT** 5, **VAL** "X" doit devenir 10 **LET** Y = **VAL** "X"
15 **PLOT** 5, Y

STR$ appliqué à un chiffre résulte dans ce qui apparaît sur l'écran si le chiffre avait été imprimé par une déclaration **PRINT**. Par ex.: **STR$** 3.5 = "3.5".

4. Comme c'est le cas pour les chiffres, il est également possible de les combiner en expressions de chaînes, comme

VAL (STR$ LEN "123456" + "-4")

ce qui est évalué comme:

VAL (STR$ LEN "123456" + "−4")

VAL (STR$ 6 + "−4")

VAL ("6" + "−4")

VAL ("6−4")

VAL "6−4"

6−4

2

Sommaire

Chaînes.
Opérations: + (pour chaînes)
Fonctions: **LEN**, **VAL**, **STR$**.

Exercices:

1. Imprimez:

LET A$ = "2 + 2"

puis:

PRINT A$; "="; **VAL** A$

Essayez de remplacer A$ par d'autres éléments plus compliqués. Par ex.:

LET A$ = "**ATN** 1*4"

(la réponse devrait être PI).

2. La chaîne "" sans symboles s'appelle une chaîne vide. C'est la seule chaîne dont la longueur est Ø. Sachez qu'un espace est significatif et qu'une chaîne vide diffère d'une chaîne contenant des espaces.

Ne la confondez pas avec le symbole de citation "" (un symbole unique, SHIFT + Q).

Celui-ci est un symbole particulier permettant de contourner l'obstacle formé par le fait qu'il est impossible d'imprimer un guillemet simple au milieu d'une chaîne (pourquoi pas?).

Quand le symbole de citation apparaît dans une chaîne ayant des guillemets à la fin (par ex. dans la liste d'un programme), il apparaît comme deux symboles de citation afin de les distinguer des guillemets ordinaires.

Mais quand il est imprimé dans une déclaration **PRINT**, il apparaît comme un simple symbole de citation.

Essayez:

PRINT "X"; ""; "X"; "" ""; "" ""; ""; "" ""

3. Si vous aimez humilier les ordinateurs, essayez:

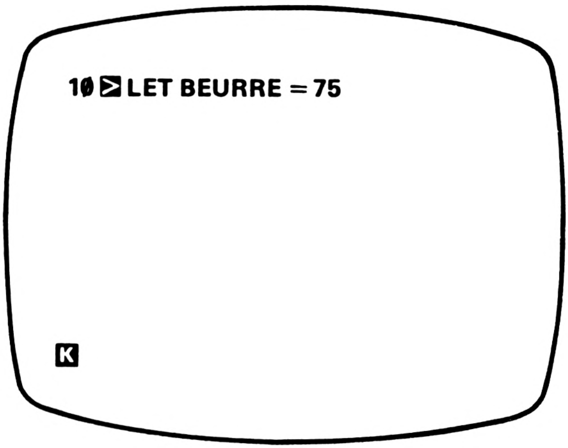
PRINT "2 + 2 ="; 2 + 1

PROGRAMMATION DE L'ORDINATEUR

Enfin vous pouvez écrire votre programme d'ordinateur. Eteignez et rallumez l'ordinateur afin d'être certain qu'il soit vide (clear). Ecrivez maintenant:

1Ø **LET** BUTTER = 75 + **NEWLINE** (butter = beurre).

et l'écran se présentera comme ci-dessous:



Ceci est différent de ce qui se passait au chapitre 6. Si vous écrivez:

PRINT BUTTER

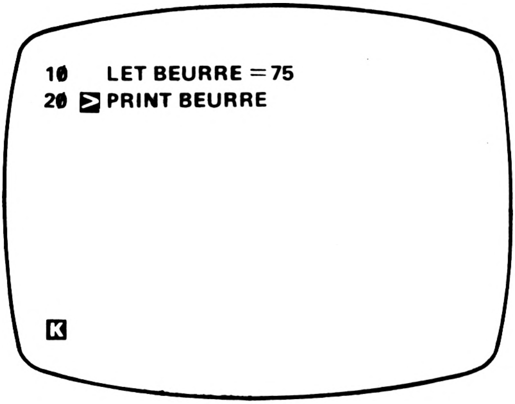
vous verrez que la variable BUTTER n'a pas été introduite. Frappez **NEWLINE** et aussitôt l'écran se présentera comme avant (cfr. le dessin).

Parce que la déclaration **LET** était précédée par le chiffre 1Ø, l'ordinateur ne l'a pas exécutée mais il l'a gardée pour plus tard. 1Ø est le numéro de la ligne et est employé de la même façon que le nom des variables, c.-à-d. comme référence.

Un ensemble de ces déclarations emmagasinées est appelé un programme. Ecrivez:

2Ø **PRINT** BUTTER

L'écran sera comme suit:



C'est la liste (imprimée) de votre programme. Pour voir votre programme exécuté, écrivez:

RUN (n'oubliez pas **NEWLINE**)

et la réponse 75 apparaîtra dans le coin gauche de l'écran. En bas, à gauche, vous apercevrez le rapport Ø/2Ø.", pas de problèmes; et 2Ø est le numéro de la ligne finale. Enfoncez **NEWLINE** et la liste reviendra sur l'écran.

Remarquez que les déclarations ont été exécutées selon l'ordre de numéros de ligne.

Supposons que vous venez de vous rappeler que vous avez oublié de mentionner le prix de yeast (la levure). Ecrivez:

15 **LET** YEAST = 40

et cela s'intercalera. Ceci n'aurait pas marché si vous aviez numéroté les deux premières lignes 1 et 2 au lieu de 1Ø et 2Ø. (Les numéros de lignes peuvent être situés entre 1 et 9999). C'est pourquoi, avant d'écrire un programme, il vaut mieux s'exercer à laisser un espace entre les lignes.

Maintenant, vous devez changer la ligne 2Ø en:

2Ø **PRINT** BUTTER, YEAST

Vous pouvez écrire entièrement ce changement, mais il y a un moyen d'utiliser à nouveau ce qui s'y trouvait déjà. Voyez-vous le petit signe ▶ à la ligne 15? C'est le curseur du programme et la ligne qu'il indique est la ligne courante.

Appuyez sur la touche ⇩ et le curseur descendra à la ligne 2Ø (⇧ le fait remonter).

Enfoncez la touche **EDIT** (SHIFT + 1) et la copie de la ligne 2Ø sera imprimée en bas de l'écran.

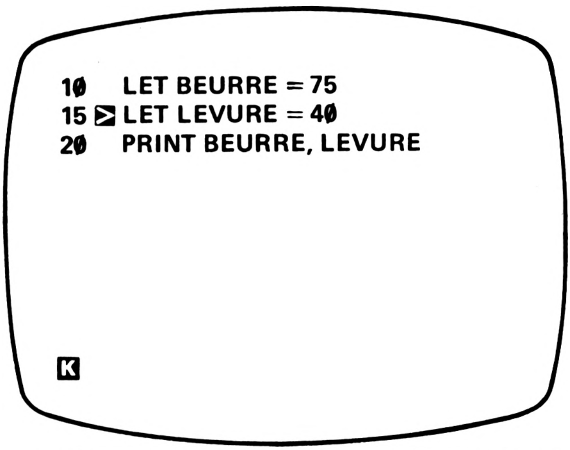
Enfoncez la touche ⇧ 7 fois afin de reculer le curseur **L** jusqu'à la fin de la ligne et écrivez:

YEAST (sans **NEWLINE**)

La ligne du fond se présente ainsi:

2Ø **PRINT** BUTTER, YEAST

Appuyez sur **NEWLINE** et l'ancienne ligne 2Ø sera remplacée. L'écran sera comme suit:



RUN (exécutez) ce programme et les deux prix seront affichés. (Voici un truc facile avec **EDIT** si vous désirez faire disparaître la partie inférieure de l'écran en même temps. Enfoncez **EDIT** et la ligne courante sera enlevée du programme, remplaçant ce que vous vouliez effacer. Si vous enfoncez **NEWLINE**, la ligne sera réintroduite dans le programme, sans aucune différence, et la partie inférieure de l'écran sera effacée, ne laissant que le curseur).
Ecrivez par inattention:

12 **LET** YEAST = 2Ø

Cette donnée sera introduite dans le programme et vous remarquerez votre erreur. Pour effacer cette ligne superflue, écrivez:

12 (avec **NEWLINE** évidemment).

Vous vous apercevrez que le curseur du programme a disparu. Imaginez-vous qu'il est caché entre la ligne 1Ø et 15, donc si vous appuyez, il remontera vers la ligne 1Ø, si par contre vous appuyez sur il descendra vers la ligne 15.
Ecrivez finalement:

LIST 15

Vous verrez apparaître sur l'écran:

15⊒ **LET** YEAST = 4Ø
2Ø **PRINT** BUTTER, YEAST

La ligne 1Ø aura disparu de l'écran mais elle reste toujours incorporée dans votre programme. Vous pouvez le contrôler en enfonçant la touche **NEWLINE**. Le seul effet de **LIST** 15 est de faire une liste, qui commence à la ligne 15, et de placer le curseur à la ligne 15.

LIST

employé seul vous montre tout le programme.

Sommaire

Programme.
Editer un programme avec: ⇧, ⇩ et **EDIT**.
Déclarations: **RUN** et **LIST**.

Exercices:

1. Modifiez le programme de telle façon qu'il ne vous montre pas seulement les deux prix mais également les messages qui permettent de savoir quoi est quoi.

2. Employez la touche **EDIT** pour changer le prix de BUTTER.

3. Exécutez le programme (**RUN**) et écrivez:
PRINT BUTTER, YEAST

Les variables restent, bien que le programme soit terminé.

4. Ecrivez:
12 (+ **NEWLINE**)

Le curseur se cache de nouveau entre la ligne 1Ø et 15. Enfoncez maintenant **EDIT** et la ligne 15 descendra. Quand le curseur est caché entre deux lignes, **EDIT** fait descendre la seconde ligne.
Frappez **NEWLINE** afin de nettoyer le bas de l'écran. Ecrivez:

3Ø

Cette fois-ci, le curseur est caché après la dernière ligne du programme. Si vous enfoncez **EDIT**, la ligne 2Ø descendra.

5. Insérez une déclaration **LIST** dans le programme pour que, quand vous l'exécutez, il s'affiche automatiquement à l'écran.

PROGRAMMATION (suite)

Ecrivez:

NEW

Ceci effacera tous les anciens programmes et variables du ZX81 (comme **CLEAR**, mais **CLEAR** n'efface que les variables).

Ecrivez maintenant dans ce programme:

10 **REM** THIS PROGRAM EXTRACTS SQUARE ROOTS
20 **INPUT** A
30 **PRINT** A, **SQR** A
40 **GOTO** 20

Vous devez écrire vous-même la plupart des espaces de la ligne 10. Exécutez maintenant le programme (**RUN**). Apparemment, l'écran reste vierge et rien ne se passe, mais regardez le curseur en bas de l'écran à gauche. Vous vous attendez à un **K**, mais vous voyez apparaître un **L**. La machine passe au mode **INPUT** (introduction). Ceci est l'effet de la déclaration **INPUT** de la ligne 20. La machine s'attend à recevoir un chiffre (ou une expression) et elle n'exécutera pas avant.
Ensuite l'effet sera:

Mais que se passe-t-il à la ligne 10? C'est comme si l'ordinateur l'ignore complètement. **REM** à la ligne 10 veut dire: Remarque ou Mémoire. Sa seule fonction est de vous rappeler ce que le programme fait. Une déclaration **REM** consiste d'un **REM** + n'importe quoi et l'ordinateur va toujours l'ignorer. Bon. Nous arrivons au mode **INPUT** de la ligne 20. Ecrivez 4 et **NEWLINE**.

Le chiffre 4 et sa racine carrée apparaissent sur l'écran. Vous pensez peut-être que c'est fini. Pas du tout. Il semble demander un autre chiffre et ceci parce que la ligne 40 **GOTO** renvoie à la ligne 20. Ainsi le programme recommence au lieu de s'arrêter. L'ordinateur fait un bond vers la ligne 20 et recommence.

Ecrivez un autre chiffre, par exemple 2. En tout cas, il vaut mieux prendre un chiffre positif. Après quelques essais, vous vous poserez la question si la machine n'en aura jamais marre. Jamais. La prochaine fois qu'elle vous demande un chiffre, donnez-lui le **STOP** (**SHIFT** + A). Elle comprendra. Vous recevrez le rapport D/20. Cherchez D dans la liste des rapports (annexe B).

20 est la ligne où l'ordinateur se trouvait quand il demanda un nouvel **INPUT** et que vous l'avez arrêté.

Est-ce que vous pouvez penser à d'autres chiffres dont vous voudriez avoir la racine carrée? Alors écrivez:

CONT (l'abbréviation de CONTINUE)

et l'ordinateur balaiera tout l'écran et vous demandera d'autres chiffres.

Pour **CONT**, l'ordinateur se souvient du dernier rapport qu'il vous a envoyé et qui avait un code différent de Ø, et il retourne vers cette ligne. Comme le dernier rapport était D/2Ø (et D n'est pas Ø), **CONT** est identique à **GOTO** 2Ø, au moins dans notre exemple. Introduisez des chiffres jusqu'à ce que l'écran soit rempli. Quand il est plein, l'ordinateur vous enverra le rapport 5/3Ø.

5 veut dire que l'écran est plein et 3Ø est le numéro de la déclaration **PRINT** qu'il essayait d'exécuter au moment où il découvrait qu'il n'y avait plus de place. De nouveau:

CONT

balaiera l'écran et continue. Cette fois-ci, **CONT** veut dire: **GOTO** 3Ø.

Notez que l'écran n'est pas vide parce qu'il s'agit d'une déclaration, mais par contre parce qu'il s'agit d'une instruction. Toutes les instructions (excepté **COPY**, voir chapitre 2Ø) vident l'écran.

Quand tout cela vous fatigue, arrêtez le programme par **STOP** et reprenez la liste avec **NEWLINE**.

Regardez la déclaration **PRINT** de la ligne 3Ø. Chaque fois que l'ensemble de chiffres A et **SQR** A est imprimé, c'est sur une nouvelle ligne parce que la déclaration **PRINT** ne se termine pas par une virgule ou un point-virgule. Si cela n'est pas le cas, la prochaine déclaration **PRINT** commence par une nouvelle ligne (donc si vous voulez incorporer une ligne blanco, utilisez la déclaration **PRINT** là où il n'y a rien à imprimer – rien que **PRINT**).

Néanmoins, une déclaration **PRINT** peut se terminer par une virgule ou un point-virgule. Alors la prochaine déclaration **PRINT** continue à imprimer comme si les deux n'étaient qu'une seule et longue déclaration.

Par ex. avec virgules, remplacez le 3Ø par:

3Ø **PRINT** A,

et exécutez (**RUN**) le programme. Vous verrez comment les déclarations **PRINT** successives peuvent imprimer sur la même ligne mais en deux colonnes.

Avec des points-virgules d'autre part:

3Ø **PRINT** A;

Tout est entassé.

Essayez:

3Ø **PRINT** A

Ecrivez les lignes suivantes:

```
1ØØ REM THIS PROGRAM MEASURES STRINGS
11Ø INPUT A$
12Ø PRINT A$, LEN A$
13Ø GOTO 11Ø
```

Ceci est un programme séparé du premier mais vous pouvez les garder en même temps. Pour exécuter le nouveau programme, écrivez:

RUN 1ØØ

Ce programme introduit une chaîne au lieu d'un chiffre et l'exprime en toute sa longueur.

CAT (et **NEWLINE**)

L'ordinateur vous donne deux guillemets parce qu'il s'attend à une chaîne.

Ceci est un aide-mémoire et en plus, vous aurez moins à écrire.

Mais vous ne devez pas vous restreindre à la constante de la chaîne (les chaînes explicites avec leurs guillemets au début et à la fin et avec tous les caractères entre les deux). L'ordinateur évaluera les expressions de chaîne, même celles avec des variables de chaîne. Dans ce cas, vous devez probablement effacer les guillemets que l'ordinateur vous a proposés. Essayez ceci: effacez-les (avec ⇨ et **RUBOUT** deux fois) et écrivez:

A$

Puisque A$ a toujours la valeur "CAT" (chat), la réponse est à nouveau CAT 3.

Ecrivez maintenant:

A$

mais cette fois-ci, sans effacer les guillemets de la chaîne. Maintenant A$ a la valeur de "A$" et la réponse est 2.

Si vous employez **STOP** comme in put de la chaîne, il faut d'abord déplacer le curseur vers le début de la ligne avec ⇦ .

Maintenant souvenez-vous de **RUN** 1ØØ que nous avons mentionné ci-dessus. Il saute à la ligne 1ØØ. Est-ce qu'on pourrait employer **GOTO** 1ØØ à la place?

Dans ce cas-ci, la réponse est positive. Mais il y a tout de même une différence. **RUN** 1ØØ efface les variables (comme **CLEAR** au chapitre 6) et pour le reste, il a la même fonction que **GOTO** 1ØØ. **GOTO** 1ØØ n'efface rien du tout. Il y aura des occasions où vous aimeriez exécuter un programme sans effacer les variables. Dans des cas pareils, utilisez **GOTO**, **RUN** aurait un effet désastreux. Ne prenez donc jamais l'habitude d'écrire **RUN** automatiquement pour exécuter un programme.

Il y a encore une autre différence. On peut écrire **RUN** sans numéro de ligne et l'ordinateur part de la première ligne du programme. **GOTO** doit nécessairement avoir un numéro de ligne.

Les deux programmes se sont arrêtés parce que vous avez introduit **STOP** à la ligne **INPUT**. Parfois, par erreur, vous écrivez un programme qui ne peut pas être arrêté et il ne s'arrêtera pas tout seul. Ecrivez:

2ØØ **GOTO** 2ØØ
RUN 2ØØ

A première vue, cela pourrait continuer jusqu'à ce que vous débranchiez l'ordinateur. Mais il existe un moyen moins drastique. Appuyez sur la touche **SPACE** au-dessus de laquelle est mentionné **BREAK**. Le programme s'arrêtera avec la mention D/2ØØ.

A la fin de chaque ligne du programme, l'ordinateur vérifie si la touche a été employée; si tel est le cas, il s'arrête. La touche **BREAK** est également utilisée quand l'enregistreur ou l'imprimante fonctionnent.

Nous connaissons déjà les déclarations **PRINT, LET, INPUT, RUN, LIST, GOTO, CONT, CLEAR, NEW, REM** et la plupart peuvent être utilisées comme instruction ou comme ligne de programme. Ceci vaut pour pratiquement toutes les déclarations du **BASIC**.

La seule vraie exception est **INPUT**, qui ne peut pas être employée comme instruction (vous recevez le rapport 8 si vous l'essayez. La raison est qu'à l'intérieur de l'ordinateur, le même espace est utilisé à la fois pour les instructions et pour les données introduites, de sorte qu'une instruction **INPUT** causerait du dérangement. **RUN, LIST, CONT, CLEAR** et **NEW** ne sont pas fréquemment utilisés dans un programme, mais ils peuvent l'être.

Sommaire

Déclarations: **GOTO, CONT, INPUT, NEW, REM, PRINT.**
STOP: comme données d'introduction.
BREAK.

Exercices

1. Essayez de remplacer la ligne 4Ø dans le programme de la racine carrée par **GOTO** 5, **GOTO** 1Ø, **GOTO** 15. Il ne devrait pas y avoir de différence significative dans le programme. Si le numéro de ligne mentionné dans une déclaration **GOTO** n'existe pas, le programme continue avec la ligne qui suit immédiatement.
Cela vaut également pour **RUN**. En effet, **RUN** considéré séparément signifie **RUN** Ø.

2. Exécutez un programme de longueur de chaîne (**LEN**) et quand il vous demande un input, écrivez:

X$ (après avoir enlevé les guillemets)

Evidemment X$ est une variable non définie et vous recevez le rapport 2/11Ø.
Si vous écrivez maintenant

LET X$ = "SOMETHING DEFINITE"

(qui a son propre rapport Ø/Ø) et

CONT

vous verrez que vous pouvez employer X$ comme donnée input (d'introduction) sans aucune difficulté. L'important c'est que **CONT** a le même effet que **GOTO** 1ØØ.

Il ne tient pas compte du rapport Ø/Ø, parce qu'il avait le code Ø et il prend le numéro de ligne du précédent rapport, 2/11Ø. Cela peut être très utile.

Si un programme s'arrête à cause d'une erreur, vous pouvez faire un tas de choses pour la corriger, et **CONT** fonctionnera toujours après.

3. Essayez le programme suivant:

1Ø **INPUT** A$
2Ø **PRINT** A$; "="; **VAL** A$
3Ø **GOTO** 1Ø

(Cfr. chapitre 7, ex. 1).

Introduisez quelques déclarations supplémentaires de print de telle façon que l'ordinateur annonce ce qu'il va faire et demande la chaîne d'input avec amabilité.

4. Ecrivez un programme pour introduire des prix et imprimez un tarif de TVA (15%). Ajoutez à nouveau des déclarations **PRINT** pour que l'ordinateur raconte ce qu'il fait. Modifiez le programme afin d'être capable d'introduire également les pourcentages de TVA.

5. Ecrivez un programme qui vous permet d'imprimer un total accumulé de chiffres que vous introduisez. (Suggestion: Prenez deux variables **TOTAL**-partons de Ø et **ITEM**.

Introduisez (input) **ITEM** et ajoutez-le à **TOTAL**; imprimez-les (**PRINT**) et recommencez).

6. Les listages automatiques (ceux qui ne résultent pas d'une déclaration **LIST**) vous ont peut-être étonné. Si vous écrivez un programme de 5Ø lignes – rien que des déclarations **REM**

1 **REM**
2 **REM**
3 **REM**
⋮ ⋮
49 **REM**
5Ø **REM**

vous serez capable d'expérimenter.

La première chose à retenir est que la ligne courante (avec ■) apparaîtra toujours sur l'écran, et de préférence au milieu.

LIST (et **NEWLINE** évidemment).

Enfoncez encore une fois la touche **NEWLINE** et vous verrez sur l'écran les lignes 1 à 22. Ecrivez maintenant:

23 **REM**

et vous devez voir les lignes 2 à 23 sur l'écran. Ecrivez:

28 **REM**

et vous aurez les lignes 27 à 48 (Dans les deux cas, en écrivant sur une nouvelle ligne, vous avez déplacé le curseur de façon à ce qu'une nouvelle liste a été conçue). Est-ce que cela paraît arbitraire? Pourtant l'ordinateur essaye de vous donner exactement ce que vous voulez; mais puisque l'être humain est imprévisible, il ne devine pas toujours exactement vos intentions.

L'ordinateur n'enregistre pas seulement la ligne courante mais également la première ligne de l'écran. Quand il essaye de faire une liste, la première chose qu'il fait est de comparer la ligne courante à la première ligne.

Si la première ligne suit la ligne courante, il n'y a pas de raison pour commencer là, et l'ordinateur emploie la ligne courante comme une première ligne et il fait la liste. Si la ligne courante apparaît sur l'écran, tout est en ordre. Si la ligne courante est en bas de l'écran, l'ordinateur déplace la première ligne vers le bas et il recommence. Si la ligne courante n'est pas tout à fait en bas de l'écran, il change la ligne précédant le curseur en première ligne.

Essayez de déplacer la ligne courante en écrivant:

line number **REM**

LIST déplace la ligne courante, mais pas la première ligne. De ce fait, les listes suivantes peuvent être différentes. Par ex. écrivez:

LIST

pour avoir la liste **LIST** et enfoncez à nouveau **NEWLINE** pour faire de la ligne Ø la première ligne. Vous devez avoir les lignes de 1 à 22 sur votre écran. Ecrivez:

LIST 22

ce qui vous donne les lignes 22 à 43. Quand vous enfoncez **NEWLINE** encore une fois, vous retrouverez les lignes 1 à 22. Ceci est plus utile pour des programmes courts que pour des longs.

7. Que feraient **CONT, CLEAR** et **NEW** dans un programme? Pouvez-vous leur trouver une fonction?

LA DECLARATION IF...

Tous les programmes qu'on a étudiés jusqu'ici ont été faciles à prédire – ils exécutaient directement toutes les instructions et parfois ils recommençaient dès le début. Cela n'est pas tellement utile. En pratique, on s'attend à ce que l'ordinateur prenne ses décisions et les exécute. Il sait le faire grâce à la déclaration **IF**.

Effacez tout (en employant **NEW**), écrivez ce petit programme et exécutez-le.

```
 10 PRINT "SHALL I TELL YOU A JOKE?"
 20 INPUT A$
 30 IF A$ = "GET LOST" THEN GOTO 200
 40 PRINT "HOW MANY LEGS HAS A HORSE GOT?"
 50 INPUT LEGS
 60 IF LEGS = 6 THEN GOTO 100
 70 PRINT "NO, 6. FORE LEGS IN FRONT", "AND TWO BEHIND".
 80 STOP
100 PRINT "YES.",,"SHALL I TELL IT YOU AGAIN?"
110 GOTO 20
200 PRINT "ALL RIGHT, THEN, I WONT."
```

Avant de traiter de la déclaration **IF**, vous devez d'abord faire attention à la déclaration **STOP** à la ligne 80. Une déclaration **STOP** arrête le programme et donne le rapport 9.

Comme vous l'avez déjà vu, une déclaration **IF** prend la forme

IF condition **THEN** déclaration

Il s'agit ici de déclarations **GOTO**, mais vous pouvez également employer des déclarations **IF**.

La condition est quelque chose qui doit encore être prouvée correcte ou fausse. Si elle est correcte, la déclaration qui suit **THEN** est exécutée; si elle est fausse, la déclaration est omise.

Les conditions les plus utiles consistent à comparer deux chiffres ou deux chaînes: elles sont capables de contrôler si deux chiffres sont identiques, ou si un chiffre est plus grand qu'un autre; et elles peuvent tester si deux chaînes sont identiques ou si une chaîne précède une autre dans l'ordre alphabétique. Ces conditions utilisent les relations =, <, >, < =, > = et <>.

= est employé deux fois dans ce programme (1 fois pour des chiffres et 1 fois pour des chaînes) et est appelé "égalité" (est égale à). Ce symbole n'a pas la même signification que = dans une déclaration **LET**.

< veut dire "est moins que". Ainsi:

$1 < 2$
$-2 < -1$
$-3 < -1$ sont tous corrects.

Mais:

$1 < 0$
$0 < -2$ sont tous faux

Afin de connaître le fonctionnement, nous allons écrire un programme pour introduire des chiffres (input) et pour montrer lequel des chiffres est le plus grand.

```
10 PRINT "NUMBER", "BIGGEST SO FAR"
20 INPUT A
30 LET BIGGEST = A
40 PRINT A, BIGGEST
50 INPUT A
60 IF BIGGEST < A THEN LET BIGGEST = A
70 GOTO 40
```

La partie cruciale est la ligne 60 qui met à jour le chiffre le plus grand (**BIGGEST**) si sa valeur précédente est inférieure à celle du chiffre ajouté (input A).

> (**SHIFT** + M) signifie "est plus grand que" et ressemble à < à l'envers. Vous pouvez retenir la différence entre les deux, car la pointe indique le chiffre le plus petit.

<= (**SHIFT** + R - n'employez pas < suivi de =) signifie "est moins que ou égal à", c.-à-d. que ce symbole ressemble à < excepté qu'on tient également compte de chiffres identiques. Ainsi 2 <= 2 est correct, mais 2 < 2 est faux.

>= (**SHIFT** + Y) signifie "est plus grand que ou égal à" et ressemble, de la même façon, à >.

<> (**SHIFT** + T) signifie "n'est pas identique à" et est opposé à =.

Ces 6 opérations en relation mutuelle ont une priorité de 5. Les mathématiciens écrivent normalement <=, >= et <> comme ≤, ≥ et ≠ et "2 <3<4" au lieu de "2<3 et 3<4", mais cela est impossible en langage BASIC.

Ces relations peuvent être combinées en employant les opérations logiques **AND**, **OR** et **NOT** (et, ou et pas).

une relation **AND** une autre

est correcte quand les deux relations sont correctes.

une relation **OR** une autre

est correcte quand une des deux relations est correcte (ou quand les 2 sont correctes).

NOT relation

est correcte quand la relation est fausse, et est fausse quand la relation est correcte. Nous pouvons faire des expressions logiques avec des relations combinées avec **AND**, **OR** et **NOT**, comme il est possible de faire des expressions numériques avec des chiffres combinés avec +, -, etc. Il est même possible d'employer des parenthèses.

NOT a une priorité de 4

AND a une priorité de 3

OR a une priorité de 2

Puisque **NOT** (contrairement aux autres fonctions) a une priorité assez petite, son argument n'a pas besoin d'être placé entre parenthèses, sauf s'il contient **AND** ou **OR**;

ainsi **NOT** A = B veut dire la même chose que **NOT** (A = B) (ce qui est identique à A<>B).
Afin d'illustrer cette théorie, videz l'ordinateur et essayez le programme suivant

```
1Ø INPUT F$
2Ø INPUT AGE
3Ø IF F$="X" AND AGE<18 OR F$="AA" AND AGE<14
THEN PRINT
"DONT";
4Ø PRINT "LET IN."
5Ø GOTO 1Ø
```

Ici, F$ est supposé être la catégorie d'un film X pour des spectateurs de 18 ans et plus, AA pour 14 ans et plus et A ou U pour tout le monde. Le programme calcule si une personne d'un âge donné peut voir le film.

Enfin, nous pouvons non seulement comparer des chiffres, mais également des chaînes. Nous avons déjà employé "=" dans "F$ = "X"" et nous pouvons même employer les cinq autres, < etc. Qu'est-ce que "moins que" peut signifier pour des chaînes? Ce n'est surtout pas "moins longue que", ne faites donc pas d'erreurs. La définition: "une chaîne est moins qu'une autre chaîne" veut dire qu'elle vient en première place dans l'ordre alphabétique. Ainsi ces exemples sont corrects.

```
"SMITH"          < "SMYTHE"
"SMYTHE"         > "SMITH"
"BLOGGS"         < "BLOGGS-BLACKBERRY"
"BILLION"        < "MILLION"
"TCHAIKOVSKY"    < "WAGNER"
"DOLLAR"         < "POUND"
```

<= signifie "est moins que ou identique à", etc. comme pour les chiffres.
Note: dans certaines versions de BASIC, mais pas pour le BASIC du ZX81, la déclaration **IF** peut avoir la forme de

IF condition **THEN** numéro de ligne

C'est la même chose que:

IF condition **THEN GOTO** numéro de ligne

Sommaire:
Déclarations: **IF** et **STOP**.
Opérations: =, <, >, <=, >=, <>, **AND** et **OR**.
Fonction: **NOT**.

Exercices:

1. <> et = sont des opposés dans le sens que **NOT** A = B est égale à A <> B.
NOT A <> B est égale à A = B.

Assurez-vous du fait que >= est opposé à < et que <= est opposé à >.
Ainsi vous pouvez omettre **NOT** au début d'une relation en remplaçant la relation par son opposé. Ecrivez encore:

NOT (une expression logique **AND** une autre)

ceci équivaut à

NOT (la première) **OR NOT** (la seconde)

et

NOT (une expression logique **OR** une autre)

est égale à

NOT (la première) **AND NOT** (la seconde)

Ainsi il est possible d'introduire les **NOT** dans les parenthèses jusqu'à ce qu'ils s'appliquent tous à des relations, et à la fin vous les laissez tomber. **NOT** est donc une fonction inutile! Elle est, malgré tout, encore employée pour rendre un programme plus clair.

2. Parfois le BASIC emploie une autre syntaxe que l'anglais. Considérez, par exemple, la phrase anglaise: "if A doesn't equal B or C" (quand A n'est pas égal à B ou C).
Comment exprimer cette phrase en BASIC?
(La réponse n'est pas:

"**IF** A <> B **OR** C"

ni

"**IF** A <> B **OR** A <> C')

Ne vous énervez pas si vous ne comprenez pas les exemples 3, 4 ou 5; les points traités sont extrêmement raffinés.

3. Pour experts!
Essayez:

PRINT 1 = 2, 1 <> 2

Vous pourriez vous attendre à une faute de syntaxe. En fait, en ce qui concerne l'ordinateur, une valeur logique n'existe pas.

(i) =, <, >, <=, >=, et <> sont des opérations binaires avec une valeur numérique et une priorité de 5. Le résultat est 1 (quand la relation est correcte) et Ø (quand elle est fausse).

(ii) Dans:

IF condition **THEN** déclaration

la condition peut, en effet, être n'importe quelle expression numérique. Quand sa valeur est Ø, la condition peut être considérée comme étant fausse, toute autre valeur (≠ Ø) donne une condition correcte.

Ainsi la déclaration **IF** signifie:

IF condition <> Ø **THEN** déclaration

(iii) **AND**, **OR** et **NOT** sont également des opérations de valeur numérique.

X **AND** Y a la valeur de { X quand Y n'est pas zéro (correct)
Ø quand Y est zéro (faux)

X **OR** Y a la valeur de { 1 quand Y n'est pas zéro
X quand Y est zéro

NOT X a la valeur de { Ø quand Y n'est pas zéro
1 quand Y est zéro

Relisez ce chapitre à la lumière de ce qui précède et assurez-vous que tout marche.

Dans les expressions X **AND** Y, X **OR** Y et **NOT** X, X et Y prennent normalement la valeur Ø ou 1 (faux ou correct).

Calculez les 1Ø combinaisons différentes et contrôlez la fonction de **AND**, **OR**, et **NOT**.

4. Essayez ce programme-ci:

```
1Ø INPUT A
2Ø INPUT B
3Ø PRINT (A AND A>=B)+(B AND A<B)
4Ø GOTO 1Ø
```

Chaque fois, le chiffre le plus grand est imprimé. Pourquoi?
Assurez-vous du fait que:

X **AND** Y

peut signifier:

"X si Y (autrement le résultat est Ø)"

et que

X **OR** Y

peut signifier:

"X à moins que Y (dans ce cas, le résultat est 1)"

Une expression contenant **AND** et **OR** de cette manière, s'appelle une expression conditionnelle. Un exemple avec **OR** pourrait être:

LET Retail price = price less VAT*(1.15 **OR** V$ = "ZERO RATED")

Remarquez comment **AND** a tendance à se combiner avec une addition (car sa valeur subsidiaire est Ø) et **OR** avec une multiplication (car sa valeur subsidiaire est 1).

5. Il est même possible de faire des expressions conditionnelles avec une valeur de chaîne, mais seulement en utilisant **AND**.

X$ **AND** Y$ a la valeur de { X$ si Y n'est pas zéro
{ si Y est zéro

Ainsi cela signifie "X$ si Y (sinon la chaîne est vide)"

Essayez ce programme qui introduit 2 chaînes en les rangeant en ordre alphabétique.

```
10 INPUT A$
20 INPUT B$
30 IF A$<=B$ THEN GOTO 70
40 LET C$=A$
50 LET A$=B$
60 LET B$=C$
70 PRINT A$; " ";("<" AND A$<B$)+("=" AND A$=B$); " "; B$
80 GOTO 10
```

6. Essayez ce programme:

```
10 PRINT "X"
20 STOP
30 PRINT "Y"
```

En exécutant ce programme, vous verrez apparaître "X" et l'ordinateur s'arrêtera avec le rapport 9/20.

Maintenant imprimez **CONT**.

Vous pourriez vous attendre à ce que **CONT** se comporte **GOTO** 20 et que l'ordinateur s'arrête sans à nouveau montrer "Y". Mais cela ne serait pas très pratique. L'ordinateur est construit d'une telle façon qu'avec un rapport 9, (déclaration **STOP** exécutée), le numéro de ligne soit augmenté de 1 pour une déclaration **CONT.**

Ainsi, dans notre exemple, "**CONT**" se comporte comme "**GOTO** 21", ce qui se comporte comme "**GOTO** 30" puisqu'il n'y a pas de lignes entre 20 et 30.

7. Plusieurs versions de BASIC (mais pas le BASIC de ZX81) disposent d'une déclaration **ON**, ce qui prend la forme:

ON expression numérique **GOTO** numéro de ligne, numéro de ligne...

L'expression numérique est évaluée. Supposons que sa valeur soit n. Le résultat devient:

GOTO la ligne numéro n.

Par exemple:

ON A **GOTO** 100, 200, 300, 400, 500

Quand, dans cet exemple, A a la valeur de 2, **GOTO** 200 est exécuté.
Dans le BASIC du ZX81, ceci peut être remplacé par:

GOTO 100*A

Dans le cas où les numéros de lignes ne sont pas nettement des centaines, vous pourriez essayer:

GOTO une expression conditionnelle.

L'ENSEMBLE DES CARACTERES

Les lettres, chiffres, signes de ponctuation etc... qui font partie d'une chaîne s'appellent des caractères et ils forment l'alphabet ou l'ensemble des caractères qui sont employés par le ZX81. La plupart sont des symboles, mais il y en a d'autres, appelés signes, qui représentent des mots entiers comme **PRINT**, **STOP**, ** etc.

Il y a 256 caractères en tout et chacun a un code entre Ø et 255. Vous trouverez la liste complète dans l'annexe A. Pour convertir les codes en caractères, il existe 2 fonctions: **CODE** et **CHR$**.

CODE s'applique aux chaînes et donne le code du premier caractère de la chaîne

(ou Ø si la chaîne est vide).

CHR$ s'applique à un numéro et donne la chaîne à caractère unique, dont le code est dénommé par ce numéro.

Ce programme imprime l'ensemble de tous les caractères:

```
1Ø LET A = Ø
2Ø PRINT CHR$ A;
3Ø LET A = A + 1
4Ø IF A < 256 THEN GOTO 2Ø
```

En haut, vous verrez les symboles ", £, $, etc... jusqu'à Z. Ils apparaîssent tous au clavier et peuvent être employés avec le curseur **L**. Plus loin, vous verrez les mêmes caractères mais en noir sur fond blanc (vidéo inversé). Ceux-ci sont également disponibles sur le clavier. Si vous enfoncez **GRAPHICS** (SHIFT + 9), le curseur change en **G**. Ceci signifie: mode graphique. Si vous frappez un symbole, il apparaît en vidéo inversé. Ceci va continuer jusqu'à ce que vous appuyez à nouveau sur **GRAPHICS** ou **NEWLINE**. **RUBOUT** garde sa signification habituelle.

Faites attention à ne pas perdre le curseur **G** parmi tous les caractères vidéo inversés que vous venez d'imprimer.

Après vos expérimentations, il vous reste toujours un ensemble de caractères en haut. Si cela n'est pas le cas, vous devez recommencer le programme. Au début, il y a de l'espace et dix combinaisons de blanc, noir et gris. Plus loin il y a encore onze bulbes.

Ils s'appellent des symboles graphiques et ils sont employés pour dessiner.

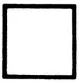
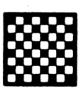
Vous pouvez les introduire en employant le clavier et le mode graphique (excepté l'espace qui est un symbole ordinaire qui utilise le curseur **L**. Le carré noir est l'espace inversé.

Employez les 2Ø touches qui ont des symboles graphiques. Par exemple, vous voulez le symbole ▞ qui se trouve sur la touche T. Appuyez sur **GRAPHICS** pour recevoir le curseur **G** et après appuyez la touche T + SHIFT. D'après les descriptions des symboles graphiques, vous vous attendez à des symboles vidéo inversés.

Mais SHIFT + T donne normalement <>, un signe, mais les signes ne s'inversent pas.

Vous recevez le symbole ▞ à la place.

Voici les 22 symboles graphiques.

Symbole	Code	Comment les obtenir	Symbole	Code	Comment les obtenir
	0	**K** or **L** **SPACE**		128	**G** **SPACE**
	1	**G** 1 en position majuscules		129	**G** Q en position majuscules
	2	**G** 2 en position majuscules		130	**G** W en position majuscules
	3	**G** 7 en position majuscules		131	**G** 6 en position majuscules
	4	**G** 4 en position majuscules		132	**G** R en position majuscules
	5	**G** 5 en position majuscules		133	**G** 8 en position majuscules
	6	**G** T en position majuscules		134	**G** Y en position majuscules
	7	**G** E en position majuscules		135	**G** 3 en position majuscules
	8	**G** A en position majuscules		136	**G** H en position majuscules
	9	**G** D en position majuscules		137	**G** G en position majuscules
	10	**G** S en position majuscules		138	**G** F en position majuscules

Regardez à nouveau l'ensemble des caractères. Il y a deux groupes distincts : un petit groupe de trois (**RND**, **INKEY$** et **PI**) après Z, et un groupe plus important qui commence avec les symboles de citation, après **Z** et qui continue de **AT** jusqu'à **COPY**. Les autres caractères semblent tous être ? En fait, ceci est la façon dont ils sont imprimés. Le vrai point d'interrogation se trouve entre : et parmi les imités, il y en a pour le contrôle des caractères comme ⇕ , **EDIT** et **NEWLINE** et le restant est pour les caractères qui n'ont aucune signification particulière pour le ZX81.

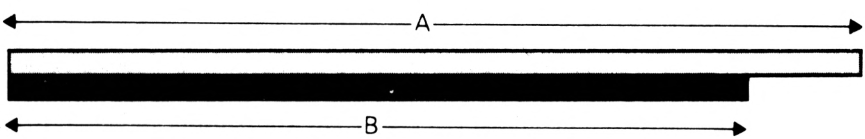
Sommaire :
Fonctions : CODE et **CHR$**.

Exercices :

1. Imaginez que l'espace pour un symbole clef est divisé en quatre parties ⊞. Si chaque partie peut être noire ou blanche, il y a 2*2*2*2 = 16 possibilités. Trouvez-les toutes dans l'ensemble des caractères.

2. Imaginez que l'espace d'un symbole est divisé horizontalement en deux ⊟. Si chaque partie peut être indifféremment noire, blanche ou grise, il y a 3*3 = 9 possibilités. Trouvez-les toutes.

3. Les caractères de l'exercice 2 sont conçus afin de les employer en barres horizontales de deux couleurs, gris et noir. Ecrivez un programme qui introduit 2 numéros A et B (situés entre Ø et 32) et qui dessine un graphique de barres.



Vous commencerez par imprimer "▓", puis vous y substituez "▀" ou "▄", selon que A est plus ou moins que B. Que fait le programme si A et B ne sont pas des nombres entiers, ou s'ils ne sont pas compris entre Ø et 32 ? Un bon programme fera quelque chose d'intelligent, quelque chose d'utile.

4. Il y a deux caractères gris sur le clavier, sur A et H. Si vous regardez attentivement, vous verrez que le caractère H ressemble à un échiquier et que le caractère A ressemble à un échiquier sur son côté. Imprimez-les l'un après l'autre et vous verrez qu'ils ne se rejoignent pas tout à fait. Le caractère sur A est fait pour se joindre à ▀ et ▄, (sur S et D), le caractère sur H est fait pour se joindre à ▌ et ▐ (sur F et G).

5. Exécutez ce programme:

```
10 INPUTA
20 PRINT CHR$ A;
30 GOTO 10
```

Si vous expérimentez, vous remarquez que pour **CHR$**, A est arrondi vers le chiffre entier le plus proche; et si A n'est pas compris entre Ø et 255, le programme s'arrête avec le rapport B.

6. Si vous employez les codes au lieu des caractères, vous pouvez étendre le concept de "classement alphabétique" aux chaînes qui utilisent n'importe quels caractères et donc pas seulement des lettres. Si, au lieu de penser en termes d'alphabet usuel de 26 lettres, vous utiliseriez un alphabet de 255 caractères, dans le même ordre de leur code, le principe reste identique. Les chaînes de l'exemple suivant sont dans l'ordre alphabétique du ZX81.

```
"   ZACHARY"
"▘"
"(ASIDE)"
"123 TAXI SERVICE"
"AASVOGEL"
"AA RDVARK"
"ZACHARY"
"AA RDVARK"
```

Voici la règle. Comparez d'abord les premiers caractères des deux chaînes. S'ils sont différents, un des deux a un code inférieur à l'autre. La chaîne à laquelle appartient le premier caractère est la première ou la moindre des deux. Si le premier caractère des deux chaînes est le même, comparez alors les caractères qui suivent. Si, selon ce procédé, une chaîne est plus courte qu'une autre, celle-ci vient en première place; sinon elles sont égales.

Réécrivez le programme de l'exercice 4 du chapitre 10 (le programme qui introduit 2 chaînes et qui les exécute ensuite) et expérimentez.

7. Ce programme remplit l'écran de caractères graphiques noirs et blanc, choisis au hasard:

```
10 LET A = INT (16 RND)
20 IF A = 8 THEN LET A = A + 120
30 PRINT CHR$ A;
40 GOTO 10
```

Comment fonctionne-t-il?

LES BOUCLES (LOOPINGS)

Supposons que vous voulez introduire 5 chiffres pour les additionner. Une façon de le faire est: (mais ne le faites pas, sauf si vous le croyez vraiment nécessaire).

```
 10 LET TOTAL=0
 20 INPUT A
 30 LET TOTAL=TOTAL+A
 40 INPUT A
 50 LET TOTAL=TOTAL+A
 60 INPUT A
 70 LET TOTAL=TOTAL+A
 80 INPUT A
 90 LET TOTAL=TOTAL+A
100 INPUT A
110 LET TOTAL=TOTAL+A
120 PRINT TOTAL
```

Cette méthode n'est pas un bon exercice de programmation. Elle reste contrôlable jusqu'à environ 5 chiffres, mais imaginez la complexité d'un programme pour additionner 10 chiffres, et additionner 100 chiffres devient vraiment impossible.
Il vaut mieux prendre une variable qui compte jusqu'à 5 et d'arrêter alors le programme; comme suit:
(vous devez vraiment faire cet exercice-ci).

```
10 LET TOTAL = 0
20 LET COMPTE = 1
30 INPUT A
40 REM COMPTE = NOMBRE D'ENTREES EFFECTIVES DE A
50 LET TOTAL = TOTAL + A
60 LET COMPTE + COMPTE + 1
70 IF COMPTE <= 5 THEN GOTO 30
80 PRINT TOTAL
```

Remarquez la facilité avec laquelle vous pouvez changer la ligne 70, pour additionner 10 ou même 100 chiffres.
Cette façon de compter est tellement utile, qu'il existe encore 2 déclarations pour faciliter le programme: les déclarations **FOR** et **NEXT**.

Elles sont toujours employées ensemble. En les utilisant, le programme introduit fait exactement la même chose que:

```
10 LET TOTAL = 0
20 FOR C=1 TO 5
30 INPUT A
40 REM C = NOMBRE D'ENTREES EFFECTIVES DE A
50 LET TOTAL = TOTAL + A
60 NEXT C
80 PRINT TOTAL
```

(Afin de remplacer le programme précédent par ce programme-ci, il suffit d'imprimer les lignes 20, 40, 60 et 70. **TO** est **SHIFT** + 4.

Remarquez que nous avons remplacé **COUNT** par C. La variable de calcul — ou variable de contrôle — d'une boucle d'itération (loop) **FOR** et **NEXT** doit avoir une seule lettre comme nom.

Le résultat d'un tel programme est que C parcourt toutes les valeurs, 1 (valeur initiale), 2, 3, 4, 5 (la limite). Pour toutes ces valeurs, les lignes 30, 40 et 50 sont exécutées. Quand C a fini avec ces 5 valeurs, la ligne 80 est exécutée.

Une subtilité en plus est que la variable de contrôle n'augmente pas nécessairement de 1; il est possible de remplacer ce 1 par n'importe quel chiffre en intercalant une partie **STEP** dans la déclaration **FOR**.

La forme usuelle d'une déclaration **FOR** est:

FOR variable de contrôle = valeur initiale **TO** limite **STEP** phase (step)

dans laquelle la variable de contrôle est une lettre, et la valeur initiale, la limite et la phase sont des expressions numériques.

Ainsi, si vous remplacez la ligne 20 par:

```
20 FOR C = 2 TO 5 STEP 3/2
```

C parcourra les valeurs 1, 2, 5 et 4. Remarquez que vous ne devez pas vous restreindre aux chiffres entiers et que la valeur de contrôle ne doit pas nécessairement atteindre la limite. Elle continue à itérer sur la suite des opérations (looping) aussi longtemps qu'elle est égale ou inférieur à la limite. (Cfr. exercice 4). Vous devez faire attention quand vous exécutez à la fois deux boucles **FOR** - **NEXT** l'un dans l'autre.

Essayez ce programme qui imprime tout un ensemble de dominos à 6 points.

```
10 FOR M=0 TO 6
20 FOR N=0 TO M              } boucle N
30 PRINT M;":";N;" ";        }              } boucle M
40 NEXT N
50 PRINT
60 NEXT M
```

Vous voyez que la boucle N est entièrement intercalée dans la boucle M; en fait, elles sont insérées l'une dans l'autre. Ce qui est à éviter, c'est d'écrire 2 boucles **FOR – NEXT** qui ne sont pas vraiment l'un dans l'autre.
Exemple:

```
FAUTIF   10 FOR M=0 TO 6
         20 FOR N=0 TO M          } boucle M
         30 PRINT M;":";N;" ";    }
         40 NEXT M                             } boucle N
         50 PRINT
         60 NEXT N
```

Quand il y a deux boucles **FOR – NEXT**, l'une doit être placée dans l'autre, ou bien elles doivent apparaître complètement séparées.

Une autre faute à éviter: pénétrer de l'extérieur dans une boucle **FOR – NEXT**. La variable de contrôle n'est imprimée qu'après l'exécution de la déclaration **FOR** et quand vous l'oubliez, l'ordinateur sera troublé par la déclaration **NEXT**.

Il est possible que l'ordinateur montre le rapport 1 (faute), ou 2 (c.-à-d. que la déclaration **NEXT** ne contient pas une variable de contrôle reconnue).
Alors vous avez encore de la chance!

Sommaire
Déclarations: **For** et **NEXT**.
TO et **STEP**.

Exercices:

1. Réécrivez le programme du chapitre 11 qui imprime l'ensemble des caractères, en employant une boucle **FOR - NEXT**. (réponse cfr. chapitre 13).

2. Une variable de contrôle n'a pas seulement un nom et une valeur, comme une variable ordinaire, mais elle a également une limite (**TO**), une phase (**STEP**) et un numéro de ligne afin de pouvoir itérer (vers la ligne suivant la déclaration **FOR**).
Il faut assurer du fait que:
 Primo: au moment où la déclaration **FOR** est exécutée, toute l'information devienne disponible.
 Secundo: cette information soit suffisante pour convertir une seule ligne

NEXT C

en deux lignes

LET C = C + 1
IF C = 5 **THEN GOTO** 3Ø

En fait, nous avons quelque peu triché: nous devrions écrire **GOTO** 21 au lieu de **GOTO** 3Ø. Mais l'effet serait identique dans notre programme.

3. Exécutez le 3ième programme et imprimez:

PRINT C

Pourquoi est-ce que la réponse est 6 et pas 5?
(réponse: la déclaration à la ligne 6Ø est exécutée 5 fois et chaque fois on ajoute 1 à C).
Qu'est-ce qui se passe si vous ajoutez **STEP** 2 à la ligne 2Ø?

4. Changez le 3ième programme pour qu'il vous demande chaque fois combien de chiffres vous voulez ajouter au lieu d'y ajouter chaque fois 5. Quand vous exécutez ce programme, qu'est-ce qui se passerait si vous introduisez Ø, c.-à-d. si vous ne voulez plus ajouter de chiffres? Pourquoi est-ce que cela pourrait troubler l'ordinateur, même quand ce message est bien clair. (L'ordinateur doit alors chercher la déclaration **NEXT** C, ce qu'il ne doit normalement pas faire).
En fait, tout ceci a été prévu.

5. Essayez ce programme pour imprimer les chiffres de 1 à 1Ø dans l'ordre inverse.

```
1Ø FOR N = 1Ø TO 1 STEP -1
2Ø PRINT N
3Ø NEXT N
```

Essayez de le remplacer par un programme sans boucles **FOR-NEXT**. Il faut s'y prendre de la même manière que dans l'exercice 2.
Pourquoi est-ce que la phase négative (**STEP**) change légèrement le tout?

RAPIDE ET LENT

Le ZX81 a deux vitesses: une rapide et une lente. En allumant l'ordinateur, il se met en mode lent et il peut traiter et montrer l'information simultanément.

Cette vitesse est idéale pour un affichage animé.

Néanmoins, il peut aller quatre fois plus vite en oubliant l'affichage, sauf s'il n'a rien d'autre à faire. Vérifiez et écrivez:

FAST

Chaque fois que vous appuyerez maintenant sur une touche, l'écran clignotera parce que l'ordinateur ne donne plus d'image pendant qu'il s'occupe de la touche que vous venez d'enfoncer.

Ecrivez un programme. Disons:

```
10 FOR N = 0 TO 255
20 PRINT CHR$ N;
30 NEXT N
```

Pendant l'exécution, l'écran sera d'un gris indéterminé jusqu'à la fin du programme quand le output (résultat) apparaît sur l'écran.

L'écran montre également une image pendant une déclaration **INPUT**, pendant que l'ordinateur attend que vous lui donniez une donnée **INPUT**.

Essayez ce programme:

```
10 INPUT A
20 PRINT A
30 GOTO 10
```

Pour revenir au mode normal (exécution et affichage), écrivez:

SLOW

Souvent, le choix entre le mode "affichage et exécution" (pour la netteté), et le mode rapide (pour la vitesse), est une question de goût.

Mais en général, vous utiliserez le mode rapide quand:

(i) Votre programme contient un tas de calculs numériques, surtout quand il n'imprime pas beaucoup. Mais le temps vous paraîtra moins long si, en utilisant le mode "exécution et affichage", les résultats apparaissent régulièrement à l'écran.

(ii) Vous écrivez un long programme. Vous avez déjà remarqué que chaque fois que vous introduisez une nouvelle ligne dans votre programme, les listes changent et à la fin, cela devient fatigant.

Dans vos programmes, vous pouvez utiliser des déclarations rapides ou lentes sans aucun problème.
Par exemple:

```
10 SLOW
20 FOR N = 1 TO 64
30 PRINT "A"
40 IF N = 32 THEN FAST
50 NEXT N
60 GOTO 10
```

Sommaire:
Déclarations: **FAST** et **SLOW**.

LES SOUS-PROGRAMMES

Parfois, des parties de votre programme auront le même travail à faire, et vous serez obligés de les imprimer plusieurs fois. Il y a moyen d'éviter cela.

Vous pouvez imprimer ces lignes une fois dans une forme dite "sous-programme" et les rappeler par après là où vous en avez besoin dans votre programme, sans être obligé de les imprimer à nouveau.

A cette fin, vous utilisez la déclaration **GOSUB** (**GO** to **SUB**routine) et **RETURN**.

GOSUB n

dans laquelle n indique le numéro de la première ligne du sous-programme, ressemble à **GOTO** n, sauf que l'ordinateur retient le numéro de ligne de la déclaration **GOSUB** de sorte qu'il puisse revenir après avoir exécuté le sous-programme.

Il est capable de retenir le numéro de ligne (adresse de retour) tout en le mettant à la tête de la liste des numéros de ligne (the **GOSUB** stack = le tas **GOSUB**).

RETURN

prend le numéro de la première ligne du tas **GOSUB** et ensuite il prend la ligne suivante. En voici un premier exemple:

```
  10 PRINT "CECI EST LE PROGRAMME PRINCIPAL"
  20 GOSUB 1000
  30 PRINT "ENCORE"
  40 GOSUB 1000
  50 PRINT "C'EST TOUT"
  60 STOP
1000 REM "LE SOUS-PROGRAMME COMMENCE ICI"
1010 PRINT "CECI EST LE SOUS-PROGRAMME"
1020 RETURN
```

La déclaration **STOP** à la ligne 60 est d'une très grande importance puisque, sans elle, le programme continuera à exécuter le sous-programme et causera l'erreur 7 au moment où la déclaration **RETURN** est atteinte.

Pour donner un exemple moins signifiant, supposons que vous voulez écrire un programme concernant des pounds, shillings et pence. (Ceux qui ont une bonne mémoire se souviendront qu'avant '71 une livre (pound) contenait 20 shillings - donc 1 shilling vaut 5 p - et 1 shilling était subdivisé en 12 anciens pence; d était l'abréviation d'un ancien penny).

Vous avez 3 variables: L, S et D (et peut-être d'autres encore: L1, S1, et D1) et l'arithmétique est très simple.

D'abord, vous calculez séparément les livres, shillings et pence – par ex. pour additionner 2 sommes d'argent, vous additionnez les livres, les shillings et les pence; pour doubler des sommes d'argent, vous doublez les livres, les shillings et les pence, etc...

Quand tout est calculé, ajustez vos calculs à la forme correcte de sorte que les pence soient situés entre Ø et 11, les shillings entre Ø et 19.

Cette dernière étape est commune à toutes les opérations de sorte que nous puissions la transformer en sous-programme.

En mettant de côté la notion de sous-programme pour un instant, ça vaut la peine d'essayer d'écrire le programme vous-même.

Comment est-ce que vous allez remplacer les données L, S et D par des pence, shillings et pounds? Mais le problème est que vous commencerez à imaginer des cas de plus en plus bizarres. Vous penserez, par exemple, à quelque chose comme 1£...25s...17d, que vous voulez convertir en 2£...6s...5d.
Ce n'est pas tellement difficile.
Mais supposez que vous avez affaire à des chiffres négatifs! Et que faire avec les fractions?
1£...25s...17d divisé par deux donne 0.5£...12.5s...8.5d et quoique les pence soient situés entre Ø et 11, les shillings entre Ø et 19, ce résultat est moins bon que 1£...3s...2.5d.
Essayez de trouver vous-même les bonnes solutions – et mettez-les dans un programme – avant de lire la suite.
Voici une des solutions possibles:

```
1ØØØ REM SOUS-PROGRAMME D'AJUSTEMENT DES L,S,D EN
     FORME NORMALE POUR LES LIVRES, SHILLINGS ET PENCE
1Ø1Ø LET D=24Ø*L+12*S+D
1Ø2Ø REM MAINTENANT TOUT LE MONTANT EST EN PENCE
1Ø3Ø LET E=SGN D
1Ø4Ø LET D=ABS D
1Ø5Ø REM NOUS TRAVAILLONS SUR D POSITIF, EN CONSERVANT
     SON SIGNE DANS E
1Ø6Ø LET S=INT (D/12)
1Ø7Ø LET D=(D-12*S)*E
1Ø8Ø LET L=INT (S/2Ø)*E
1Ø9Ø LET S=S*E-2Ø*L
11ØØ RETURN
```

Vue à part, ce programme n'a pas beaucoup d'utilité, puisqu'on ne pourra pas s'en servir par la suite. Imprimez ce programme principal et un autre sous-programme pour imprimer L, S et D.

```
  10 INPUT L
  20 INPUT S
  30 INPUT D
  40 GOSUB 2000
  45 REM ECRIRE LES VALEURS
  50 PRINT
  60 PRINT "□□□ =";
  70 GOSUB 1000
  75 REM AJUSTEMENT
  80 GOSUB 2000
  85 REM ECRIRE LES VALEURS
  90 PRINT
 100 GOTO 10

2000 REM SOUS PROGRAMME D'ECRITURE DE L,S ET D
2010 PRINT "£";L;"..";S;"S..";D;"D";
2020 RETURN
```

(Souvenez-vous du chapitre 9: une déclaration **PRINT** qui est vide (cfr. ligne 50) imprime un espace vide).

Il est clair que nous avons économisé dans notre programme en utilisant le sous-programme **PRINT** à la ligne **2000** et c'est ainsi que les sous-programmes sont généralement employés: afin de réduire les programmes principaux.

Néanmoins, ce sous-programme allonge le programme à cause de **GOSUB** et **RETURN**.

Cela veut dire que la longueur du programme n'est pas le problème principal.

Avec une certaine adresse, vous pouvez simplifier les programmes avec l'aide des sous-programmes. Le programme principal est simplifié: chaque déclaration **GOSUB** contient du **BASIC** très compliqué, mais ne pensez pas à çà, ce n'est que le résultat qui compte. Ainsi la structure du programme principal devient plus claire.

Les sous-programmes, par contre, sont simplifiés pour une toute autre raison, c.-à-d. pour les raccourcir. Ils emploient toujours les anciennes déclarations pénibles: **LET** et **PRINT**, mais ils n'ont qu'à faire une partie du travail et ils sont, par conséquent, plus faciles à écrire.

L'adresse consiste à choisir le niveau (ou les niveaux) auquel on doit écrire les sous-programmes. Ils doivent être assez importants afin d'avoir un certain impact sur le programme principal, et d'autre part assez petits pour être faciles à écrire (au moins plus facile qu'un programme entier).

Ces exemples (pas recommandés) illustrent ce qui précède.

Primo:

```
  10 GOSUB 1000
  20 GOTO 10
1000 INPUT L
1010 INPUT S
1020 INPUT D
1030 PRINT " ";L;"..";S;"S..";D;"D";TAB 8;"=";
1040 LET D=240*L+12*S+D
   :      :
   :      :
2000 RETURN
```

et secundo:

```
  10 GOSUB 1010
  20 GOSUB 1020
  30 GOSUB 1030
  40 GOSUB 1040
  50 GOSUB 1050
   :      :
   :      :
  30 GOTO 10
1010 INPUT L
1015 RETURN
1020 INPUT S
1025 RETURN
1030 INPUT D
1035 RETURN
1040 PRINT " ";L;"..";S;"S..";D;"D";TAB 8; "= ";
1045 RETURN
1050 LET D=240*L+12*S+D
1055 RETURN
   :      :
   :      :
```

Le premier exemple avec son seul sous-programme de grande puissance et le second avec ses multiples sous-programmes très simples, constituent des cas extrêmes, mais l'un et l'autre sont fort peu utiles. Un sous-programme peut très bien faire appel à un autre ou à soi-même (un sous-programme qui fait appel à soi-même est une récurrente). N'ayez donc pas peur d'employer plusieurs spires.

Sommaire:
Déclarations: **GOSUB** et **RETURN**

Exercices:
1. Ce programme fonctionne pratiquement comme un calculateur LSD universel. Comment utiliseriez-vous ce programme?

(i) Pour convertir des pounds et nouveaux pence en pounds, shillings et pence?
(ii) Pour convertir des guinées en pounds et shillings (1 guinée = 1£...1s)
(iii) Pour trouver les fractions des pounds (par ex. le tiers d'un pound, ou d'un mark, est 6s...8d).

Insérez une ligne pour arrondir le pence vers le farthing le plus proche (un farthing = 1/4 penny).

2. Ajoutez 2 déclarations au programme:

```
4 LET ADJUST = 1000
7 LET LSD PRINT = 2000
```

et changez

GOSUB 1000 en **GOSUB ADJUST**
GOSUB 2000 en **GOSUB LSD PRINT**

Ceci marche comme vous l'aviez espéré. En réalité, le numéro de ligne d'une déclaration **GOSUB** (ou **GOTO** ou **RUN**) peut être n'importe quelle expression numérique. (ne vous attendez pas à ce que ça marche avec des ordinateurs autres que le ZX81 parce que ce n'est pas du **BASIC** standardisé).

Ces changements peuvent faire des miracles pour éclaircir vos programmes.

3. Réécrivez le programme principal pour faire quelque chose de différent, tout en employant les mêmes sous-programmes.

4. ... **GOSUB** n
... **RETURN**

Dans des lignes consécutives, peut être remplacé par

... **GOTO** n

Pourquoi?

5. Un sous-programme peut avoir plusieurs points d'entrée. Par exemple: en raison de la façon dont le programme principal fait usage des deux sous-programmes, avec **GOSUB** 1000 immédiatement suivi de **GOSUB** 2000, on peut les remplacer par un seul sous-programme qui ajuste L, S, et D et qui les imprime.

Il y a deux points d'entrée, un au début pour tout le sous-programme et un plus loin pour l'affichage. Faites les arrangements nécessaires.

6. Exécutez ce programme-ci:

```
10 GOSUB 20
20 GOSUB 10
```

Les adresses de retour sont entassées sur le paquet **GOSUB**, mais elles n'en sont jamais enlevées et à la fin elles encombrent l'ordinateur.
Le programme s'arrête avec le rapport **4** (annexe B).
Vous aurez certainement des difficultés à les élaguer sans les perdre toutes, mais ça ira tout de même.

(i) Eliminez les deux déclarations **GOSUB**.
(ii) Insérez deux nouvelles lignes:

```
11 RETURN
21 RETURN
```

(iii) Ecrivez:

RETURN

Les adresses de retour seront éliminées jusqu'à ce que l'erreur 7 apparaisse.
(iv) Changez votre programme afin d'éviter la même erreur.
Comment le feriez-vous?

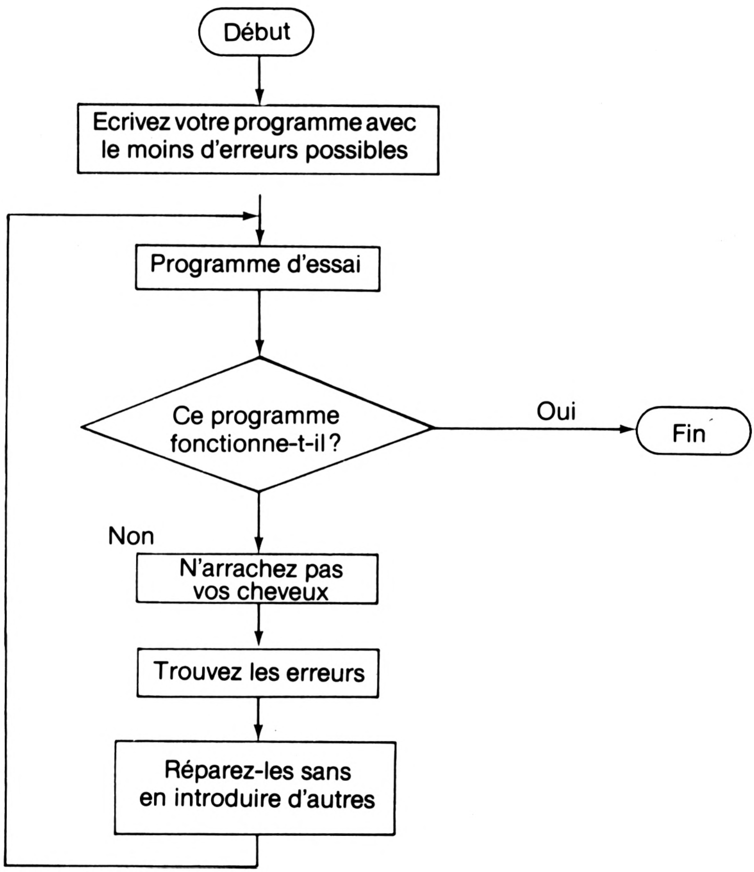
FAITES FONCTIONNER VOS PROGRAMMES

L'art de la programmation demande davantage de connaissances que celle de l'effet des déclarations. Vous avez probablement remarqué que vos programmes ont des défauts (bugs) quand vous les imprimez pour la première fois. Ce sont peut-être des fautes de frappe ou des erreurs de conception de programme. Vous pourriez imputer ces erreurs à l'inexpérience; mais ce n'est pas tout à fait ça.

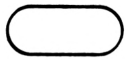
Chaque programme commence par une erreur (bug), et la plupart des programmes finissent également par une erreur. Il en résulte deux choses importantes.

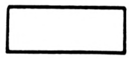
D'abord, il faut essayer les programmes tout de suite, ensuite, vous ne devez pas vous énerver chaque fois que les programmes ne marchent pas.

Le plan général peut être illustré par l'organigramme suivant.



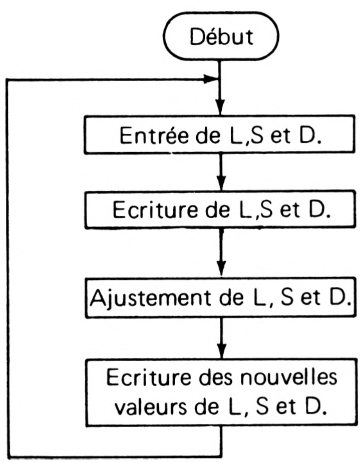
L'idée consiste à suivre les flèches de cadre en cadre en exécutant ce qu'ils vous demandent. Nous avons employé des cadres différents pour des instructions différentes.

Le cadre rond:  signifie: départ ou fin

Le cadre rectangulaire:  signifie: continuez tout droit

Le losange:  vous demande de prendre une décision avant de continuer.

(Ces figures sont des symboles universels, mais elles n'ont pas une trop grande importance). Evidemment, ces organigrammes sont mal adaptés à la description de l'activité humaine. La pensée qui suit des directives rigides, n'est ni flexible ni créative. Néanmoins, c'est ce qu'il faut pour les ordinateurs. Les organigrammes sont idéaux pour décrire – en gros – la structure des programmes, avec un sous-programme dans presque chaque cadre. Un organigramme pour notre programme de livre sterling pourrait se présenter comme suit:



Les cadres rectangulaires représentent des **GOSUB**. Néanmoins, en pratique, il y a des cadres (comme par ex. le cadres "INPUT" qui introduit L, S et D) qui sont immédiatement traduits en BASIC, sans sous-programmes.

Tout ce qui rend le programme plus clair, comme les organigrammes, les sous-programmes et les déclarations **REM**, vous aide à mieux le comprendre. Ainsi, vous éviterez bien des erreurs. Les sous-programmes peuvent également vous aider à sortir les erreurs que vous avez déjà écrites, en rendant le programme plus facile à contrôler. Il vous sera plus facile de contrôler les sous-programmes un par un, et de vous assurer qu'ils fonctionnent, que de contrôler un programme entier et mal structuré.

Les sous-programmes vous aident avec le cadre "trouvez les erreurs" (c.-à-d. le cadre qui vous demande de chercher les erreurs). C'est le cadre pour lequel vous aurez besoin de toute l'aide que vous pouvez vous procurer, puisque c'est le plus exaspérant.
Voici d'autres recommandations pour vous aider à trouver des erreurs:

(i) Vérifiez s'il n'y a pas de fautes de frappe. N'oubliez jamais de le faire.

(ii) Essayez de trouver toutes les variables à chaque étape. Expliquez-les avec des déclarations **REM**. Vous pouvez contrôler les variables à un certain point du programme, en insérant une déclaration **PRINT**.

(iii) Si le programme s'arrête avec un rapport "erreur", utilisez cette information autant que possible. Vérifiez le code d'erreur et cherchez la raison pour laquelle le programme s'arrête à telle ou telle ligne. Imprimez, si nécessaire, les valeurs des variables.

(iv) Vous pourriez être obligé de parcourir le programme ligne par ligne, en imprimant les lignes comme des commandes.

(v) Faites comme si vous étiez l'ordinateur. Prenez un crayon et du papier et exécutez le programme en notant les valeurs des variables. Une fois que vous avez trouvé les erreurs, vous pouvez les corriger dans le programme original, mais alors il est nécessaire de le contrôler à nouveau. Il est trop facile de corriger une erreur en la remplaçant par une autre.

Exercices:

1. L'organigramme du calculateur LSD n'a pas de cadre "fin". Est-ce vraiment indispensable? Où le mettriez-vous, s'il était indispensable?

2. Etablissez un organigramme pour les programmes du chapitre 12 (les programmes avec les boucles).

EMMAGASINAGE SUR BANDES MAGNETIQUES

Comme nous l'avons mentionné au chapitre 1 et comme vous l'avez déjà remarqué, vous perdez les programmes et toutes les variables qu'ils contiennent, chaque fois que vous éteignez le ZX81. La seule façon de les conserver consiste à les faire enregistrer sur bandes magnétiques par l'ordinateur. Par après, vous pouvez recharger l'ordinateur des bandes ou des cassettes et il sera remis dans son état initial, c.-à-d. l'état du moment de l'enregistrement.

Vous avez trouvé une paire de câbles dans la boîte (article 5 chapitre 1) qui connectent le ZX81 à un enregistreur. Vous devez fournir votre propre enregistreur, et il y en a qui marchent mieux que certains autres.

En premier lieu, pour ce qui est du ZX81, les enregistreurs de cassettes mono, qui sont portables et bon marché, sont au moins aussi efficaces que les chers, et en plus ils sont plus faciles à manier. Un compteur de bandes vous sera certainement très utile.

En second lieu, l'enregistreur doit avoir une prise input pour micro et une prise output pour les écouteurs. (Si elle manque, essayez la prise pour haut parleur externe.) Celles-ci doivent être, de préférence, des prises pour cavaliers de 3,5 mm (afin de s'accorder avec les cavaliers des câbles prévus), parce que les autres ne font pas passer des signaux qui sont assez puissants pour le ZX81.

Vous pouvez employer n'importe quelle sorte de cassettes, quoique les cassettes "low noise" soient préférables. Ayant acquis l'enregistreur, connectez-le à l'ordinateur. Le premier câble doit connecter la prise micro (input) à l'enregistreur au moyen de la prise **MIC** de l'ordinateur. Le second connecte la prise output des écouteurs à la prise **EAR** du ZX81.

Introduisez un programme dans l'ordinateur (disons le programme de l'ensemble des caractères du chapitre 11). Vous ferez mieux de donner un nom au programme afin d'être capable de le récupérer par après et il est préférable que ce nom soit mentionné dans le programme de façon qu'il apparaisse sur les listes. Une déclaration **REM** fait bien l'affaire. Ecrivez:

5 **REM** "CHARACTERS"

Ceci est une exécution à vide, écrivez maintenant:

SAVE "CHARACTERS"

Regardez le téléviseur. Pendant 6 secondes, vous ne verrez que du gris. Ensuite vous verrez une structure de petits traits noirs et blancs. Enfin, l'écran deviendra tout blanc avec le rapport Ø/Ø. L'ordinateur était en train d'envoyer un signal à la prise **MIC** et il envoyait également ce même message au téléviseur, ce qui produisait cette image. La partie grise représentait l'introduction silencieuse et la partie noir et blanc était le programme. Maintenant vous voulez sans doute capter le signal sur votre bande magnétique. Faisons-le correctement cette fois-ci.

MEMORISER LES PROGRAMMES

1. Placez la bande sur une partie vierge ou une partie qui peut être effacée.

2. Enregistrez le mot “characters” en le disant au micro. Ceci n'est pas essentiel mais cela vous aidera à retrouver vos programmes par après. Reconnectez l'enregistreur à l'ordinateur.

3. Ecrivez:

SAVE “CHARACTERS” (sans **NEWLINE**)

4. Actionnez l'enregistreur.

5. Appuyez **NEWLINE**

6. Regardez le téléviseur. Quand le programme est fini, arrêtez l'enregistreur. Afin d'être certain que tout a été enregistré, vous devez écouter l'enregistrement par le haut parleur de l'enregistreur. Vous devez probablement débrancher le câble de la prise des écouteurs de l'enregistreur. Rebobinez la bande et repassez-la. Ensuite il y aura un bruit sourd et doux. Ceci ne fait pas réellement partie de l'enregistrement mais c'est la fin du signal pour le téléviseur (avant d'appuyer **NEWLINE**), ce qui a également été transmis à l'enregistreur.

Ensuite il y a 5 secondes de silence qui marquent le début du signal pour l'enregistreur. Pendant ce temps, l'écran du téléviseur était gris.

Après il y a encore 6 secondes de tonalités hautes et fortes. Si vous augmentez le volume, ce bruit devient désagréablement bruyant. Ceci constitue l'enregistrement du programme et correspond aux structures noires et blanches.

Finalement le bruit sourd et doux revient.

Si vous n'avez pas entendu ce concert, vérifiez les connections de l'ordinateur et de l'enregistreur. Il arrive que le cavalier ne donne pas contact s'il n'est pas introduit assez loin dans la prise. Retirez-le de 1 ou 2 mm et vous les sentirez peut-être se mettre dans une position plus adéquate. Maintenant l'enregistrement est prêt pour l'oreille humaine et vous pouvez recharger la cassette dans l'ordinateur.

CHARGER UN PROGRAMME AVEC UN NOM

1. Rebobinez la bande jusqu'au début.

2. Assurez-vous que la prise **EAR** de l'ordinateur soit connectée à la prise des “écouteurs” de l'enregistreur.

3. Mettez le volume de l'enregistreur à 3/4 de sa puissance maximale. S'il y a un contrôle de tonalité, réglez-la de telle façon que les tons “treble” soient hauts et que les tons “bass” soient bas (jusqu'à ce que ça donne une impression enrouée).

4. Ecrivez:

LOAD "CHARACTERS" (sans **NEWLINE**)

5. Actionnez l'enregistreur.

6. Enfoncez **NEWLINE**. Vous verrez de nouveau des images de l'enregistrement sur l'écran du téléviseur, mais elles seront différentes des premières, car elles auront une structure noir et blanc.

Les deux parties, le silence et le programme, seront plus difficiles à distinguer mais vous verrez que la partie programme montre des lignes plus étroites et plus définies (essayez l'exercice 1).

Après 15 secondes, le programme doit être chargé et il s'arrête avec le rapport Ø/Ø. Sinon enfoncez la touche "**BREAK**", et tout s'arrange.

L'erreur la plus fréquente est d'avoir mal choisi le volume. Il doit être:

(i) Assez haut pour la partie programme, afin d'être entendu par l'enregistreur.
(ii) Pas trop haut pour que le programme ne soit pas distribué (ce qui ne se passe pas très souvent).
(iii) Assez bas pour que la partie "silence" soit reconnue par l'ordinateur comme telle.

Le meilleur réglage est obtenu en augmentant le volume sans pour autant distribuer la partie "silence". Faites-le en écoutant l'enregistrement à travers les écouteurs. Quand la partie "silence" reste distribuée, cela veut dire qu'il y a d'autres problèmes.

Certains enregistreurs forment une boucle de rétroaction avec le ZX81. Ce problème ne se produit que lorsque les câbles **MIC** et **EAR** sont branchés en même temps. Afin d'y remédier, il faut débrancher le câble **EAR**.

D'autres enregistreurs peuvent enregistrer un ronflement du secteur. Le remède: employez des batteries.

Encore d'autres - surtout les vieux et les usés - font des bruits de fond. Le remède à cet inconvénient est d'employer une bande de qualité supérieure, quoique ce ne soit pas vraiment nécessaire. Essayez également de nettoyer la tête de l'enregistreur si elle est sale.

Enfin il peut y avoir le même problème avec la fiche dans la prise des écouteurs, comme on a eu avec la prise du micro.

Quand vous avez un programme sur bande magnétique et vous avez oublié son nom, vous pouvez toujours le charger. (Essayez-le avec le programme de l'ensemble des caractères).

CHARGER UN PROGRAMME SANS NOM

1. Mettez la bande magnétique dans la position de l'introduction silencieuse.

2. Contrôlez et réglez les contrôles comme avant. Vous verrez que vous devriez encore davantage faire attention au niveau du volume, comparé avec un programme avec nom.

3. Imprimez:

LOAD "" (sans **NEWLINE**)

4. Mettez l'enregistreur en marche.

5. Enfoncez **NEWLINE**.

6. Le reste est comme dans l'exemple précédent. Le système est comme suit: si le nom du programme que vous voulez est la chaîne vide, l'ordinateur charge le premier programme qu'il rencontre.

Notez que, si vous mémorisez un programme, il est impossible de le nommer avec une boucle vide. Si vous l'essayez quand-même, vous verrez apparaître l'erreur F. **LOAD** (charger) et **SAVE** (mémoriser) peuvent également être employés dans des programmes.

Avec **SAVE**, le programme se conservera dans un tel état que lorsqu'il est chargé, il continue directement à exécuter la ligne suivant la déclaration **SAVE**.
Par exemple, écrivez:

```
5 REM "INUTILE"
10 PRINT "C'EST TOUT CE QU'IL FAIT"
20 STOP
100 SAVE "INUTILE"
110 GOTO 10
```

Connectez l'enregistreur et écrivez:

RUN 100 (sans **NEWLINE**)

actionnez l'enregistreur et enfoncez le bouton **NEWLINE**. Quand le programme se mémorise, il continue à marcher. Vous verrez que le dernier S de "useless" de la ligne 100 a changé en caractère inverse de video, mais cela ne fait rien.

Afin de charger la bande magnétique, rebobinez-la jusqu'avant le début du programme et écrivez:

LOAD "USELESS" (sans **NEWLINE**)

Faites passer le contenu de la bande dans l'ordinateur et enfoncez **NEWLINE**.

Quand le programme est chargé, il continue avec la ligne 110 et il exécute le tout sans aucune aide de votre part.

Notez que, en mettant la déclaration **SAVE** à la fin du programme, vous pouvez exécuter le programme sans **SAVE** en imprimant **RUN**.

Ne mémorisez jamais dans un sous-programme, cela ne marche pas très bien.

N'employez pas de caractères inverses de video pour écrire le nom d'un programme.

La partie qui suit les caractères inverses sera perdue. Un nom ne sait pas contenir plus de 127 caractères.

Le nom d'une déclaration **LOAD** ou **SAVE** ne doit pas nécessairement être une constante de chaîne, il peut être n'importe quelle expression de valeur de chaîne, comme A$ ou CHR$ 1ØØ.

Sommaire

Mémoriser un programme sur bande magnétique.
Charger un programme avec nom.
Charger le premier programme disponible d'une bande magnétique.
Mémoriser un programme de sorte qu'il se charge et s'actionne soi-même.
Déclarations: **LOAD** et **SAVE**.

Note

Il est impossible de charger des programmes mémorisés d'un autre type d'ordinateur ou d'un ZX80, avec son propre langage BASIC.

Vos programmes mémorisés ne peuvent pas être chargés dans d'autres types d'ordinateur ou dans le ZX80. Le ZX80 avec un BASIC ZX81 est néanmoins compatible avec le ZX81. Les programmes mémorisés de l'un peuvent être chargés dans l'autre.

Après avoir chargé un programme ZX80, le ZX81 travaillera à grande vitesse.

Exercices:

1. Faites une bande avec beaucoup de petits programmes. Faites entrer le contenu dans l'ordinateur et écrivez:

LOAD "NOT DE NAME OF A PROGRAMME"

Vous verrez très bien, sur l'écran, la différence entre les périodes vides (avec une structure noire et blanche assez négligée), et les programmes (lignes bien définies). Ces structures diffèrent de celles de mémorisation.

Quand vous baissez le volume pendant qu'un programme court, vous voyez que l'image change et prend la structure de l'espace vide, dès le moment où le volume devient trop faible pour ressembler à un programme.

2. Faites une bande magnétique sur laquelle le premier programme, au moment où il est chargé, imprime une liste contenant les autres programmes sur la bande (menu), demande de choisir un programme et charge le programme.

3. Imprimez à nouveau le programme des caractères et imprimez ensuite:

LET X = 7

de sorte que, même si cela n'apparaît pas dans le programme, l'ordinateur contienne une variable avec la valeur 7. Mémorisez maintenant le programme, éteignez et rallumez l'ordinateur (pour être sûr de ne pas avoir triché) et rechargez le programme.

Imprimez:

PRINT X

et vous aurez la réponse 7. La déclaration **SAVE** ne mémorise pas seulement le programme, mais également les variables, X inclus.

Quand vous voulez mémoriser les variables, tout en exécutant le programme, n'oubliez pas d'utiliser **GOTO** au lieu de **RUN**, comme mentionné au chapitre 9.

Vous pouvez éviter ce problème en employant l'auto exécuteur **SAVE** comme ligne de programme.

4. Ecrivez un programme très long et puis débranchez l'ordinateur pour un instant. Ceci peut toujours se passer spontanément; ce n'est pas une erreur (bug), mais un hasard (glitch). Il n'y a rien à faire, sauf de pleurer. Quand cela se passe un peu trop souvent, ça veut dire qu'il y a quelque chose qui cloche, mais cela vaut la peine de conserver un programme incomplet sur une bande à demi-enroulée.

IMPRIMER "A LA CARTE"

Vous vous souvenez qu'une déclaration "**PRINT**" a toute une série d'éléments qui avaient tous un sens (ou même aucun sens); ils étaient séparés d'une virgule ou d'un point-virgule. Il y a encore deux sortes de déclarations **PRINT** qui ne se réfèrent pas à ce que l'ordinateur doit faire, mais à l'endroit où il doit exécuter.

Par exemple: **PRINT AT** 11, 16 "*", imprime une étoile au milieu de l'écran.

AT ligne, colonne

déplace la déclaration **PRINT** à la ligne et à la colonne indiquées (l'endroit où le prochain élément doit être imprimé).

Les lignes sont numérotées de Ø à 21 et les colonnes de Ø à 31.

TAB colonne

déplace la déclaration à la colonne indiquée. On reste à la même ligne, mais quand ce déplacement implique un recul, on va à la ligne suivante.

Remarquez que l'ordinateur réduit le modulo des numéros de colonnes (ol divise par 32 et retient le restant), donc **TAB** 33 est égale à **TAB** 1. Par ex.:

PRINT TAB 3Ø; 1; **TAB** 12; "CONTENTS"; **AT** 3,1; "CHAPTER"; **TAB** 24; "PAGE"

(Voici un exemple d'un en-tête d'une page de registre avec le chiffre 1 comme numéro de page).

Quelques détails:

(i) Ces éléments se terminent de préférence par un point-virgule, comme on l'a fait plus haut. On peut également utiliser une virgule (ou rien du tout, à la fin d'une déclaration). Mais ceci a l'inconvénient de déplacer la position **PRINT** après l'avoir choisie consciencieusement.

(ii) Bien que **AT** et **TAB** ne soient pas des fontions, il faut pourtant utiliser la touche FONCTION (**SHIFT** + **NEWLINE**) pour les imprimer.

(iii) On ne peut pas utiliser les deux dernières lignes (22 et 23) de l'écran, car elles sont réservées aux instructions, les données **INPUT**, les rapports etc... Une référence à la dernière ligne se situe donc à la ligne 21.

(iv) On peut utiliser **AT** pour placer **PRINT** sur une ligne qui est déjà employée, l'ancien texte sera effacé.

Il y a deux autres déclarations en relation avec **PRINT**, soit **CLS** et **SCROLL**.

CLS: efface l'écran (mais rien d'autre),

SCROLL: déplace l'image d'une ligne vers le haut (vous perdez la première ligne) et déplace la position **PRINT** vers la dernière ligne.

Exécutez ce programme-ci en exemple:

```
1Ø SCROLL
2Ø INPUT A$
3Ø PRINT A$
4Ø GOTO 1Ø
```

Sommaire:
Eléments **PRINT**: **AT** et **TAB**.
Déclarations: **CLS** et **SCROLL**.

Exercices:
Exécutez le programme suivant:

```
10 FOR I = 0 TO 20
20 PRINT TAB 8*I; I;
30 NEXT I
```

Ceci montre la réduction du nombre de **TAB** modulo 32. Si vous voulez un exemple plus élégant, changez le 8 en 6 à la ligne 20.

GRAPHIQUES

Maintenant nous allons parler de quelques caractéristiques plus élégantes du ZX81. Il s'agit de ce qu'on appelle "pixels" (picture elements - les éléments d'une image).

L'écran sur lequel vous imprimez contient 22 lignes et 32 colonnes, ce qui donne 22*32 = 704 positions de caractères. Chacune de ces positions comporte 4 "pixels" divisés comme un cake.

Un "pixel" est spécifié par 2 chiffres, les coordonnées. La première est la coordonnée X, qui indique la distance entre le pixel et la colonne à l'extrême gauche (X est une croix), la seconde est la coordonnée Y, qui indique la distance entre le "pixel" et la dernière ligne.

Ces coordonnées sont écrites en paires entre parenthèses, ainsi: (Ø/Ø), (63,Ø), (Ø,43), (63,43) sont des coordonnées d'en bas à gauche, d'en bas à droite, d'en haut à gauche et d'en haut à droite.

La déclaration:

PLOT X-coordinate, Y-coordinate

noircit le pixel avec ces coordonnées,

La déclaration:

UNPLOT X-coordinate, Y-coordinate

le blanchit.

Essayez ce programme-rougeole:

```
1Ø PLOT INT (RND*64), INT (RND*44)
2Ø INPUT A$
3Ø GOTO 1Ø
```

Ce programme présente graphiquement un point choisi au hasard, chaque fois que vous enfoncez **NEWLINE**.

Maintenant un programme assez utile. Il présente graphiquement une fonction **SIN** pour des valeurs entre Ø et 2.

```
1Ø FOR N = Ø TO 63
2Ø PLOT N, 22 + 2Ø*SIN (N/32*PI)
3Ø NEXT N
```

Le programme suivant représente un graphique de **SQR** (une partie d'une parabole), entre Ø et 4.

```
1Ø FOR N = Ø TO 63
2Ø PLOT N, 2Ø*SQR (N/16)
3Ø NRXT N
```

Notez que les coordonnées d'un pixel sont assez différentes de la ligne et de la colonne d'un élément **AT**. A la fin de ce chapitre vous trouverez un diagramme qui pourrait être utile pour calculer des coordonnées d'un pixel et pour calculer des numéros de ligne et de colonne.

Exercices:

1. Il y a trois différences entre les numéros d'un élément **AT** et les coordonnées d'un pixel. Quelles sont ces différences?

Supposez qu'une position **PRINT** correspond à un élément **AT** L, C (pour ligne et colonne). Démontrez pour vous-mêmes que les 4 pixels de cette position ont des coordonnées X 2*C ou 2*C + 1, et des coordonnées Y 2*(21 - L) ou 2*(21 - L) + 1. (Regardez le diagramme).

2. Faites un programme-rongeur, en changeant le programme-rougeole. Remplissez d'abord l'écran avec des carreaux noirs (un carreau noir est un espace inversé de vidéo), et faites ensuite disparaître, au hasard, quelques points.

Si vous ne disposez que d'une mémoire de 1 K - comme toute machine standardisée sans une mémoire extra - votre réserve sera vite épuisée, de sorte que vous seriez obligés de n'employer qu'une partie de l'écran.

3. Modifiez le programme **SIN** de sorte que, avant de représenter le graphique, vous imprimez d'abord une ligne horizontale de "—", comme axe-X et une ligne verticale de "/", comme axe-Y.

4. Ecrivez des programmes pour représenter des graphiques avec plusieurs fonctions par ex. **COS, EXP, LN, ATN, INT** etc...

Pour toutes ces fonctions, vous devez contrôler si les graphiques entrent dans l'écran. Faites attention à:

(i) L'étendue des fonctions (correspondant à l'étendue de Ø à 2 π pour le graphique **SIN**).
(ii) La place qu'occupe l'axe-X sur l'écran (correspondant à 22 à la ligne 2Ø du programme **SIN**).
(iii) La gradation de l'axe-Y (correspondant à 2Ø à la ligne 2Ø du programme **SIN**).

Vous verrez que la fonction **COS** est la plus simple, elle ressemble à **SIN**.

5. Exécutez:

```
10 PLOT 21, 21
20 PRINT "HEAVY QUOTES"
30 PLOT 46, 21
```

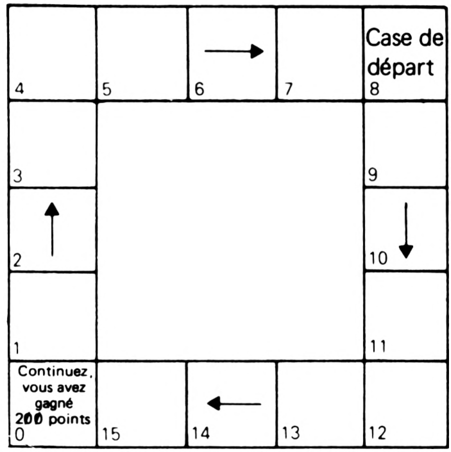
PLOT fait avancer la position **PRINT** (**UNPLOT** aussi).

6. Ce sous-programme trace une ligne assez droite du pixel A, B vers le pixel C,D. Employez-le comme partie d'un programme principal, qui fournit les valeurs A, B, C et D.

(Si vous ne disposez pas d'une extension de mémoire, vous devriez probablement omettre les déclarations **REM**).

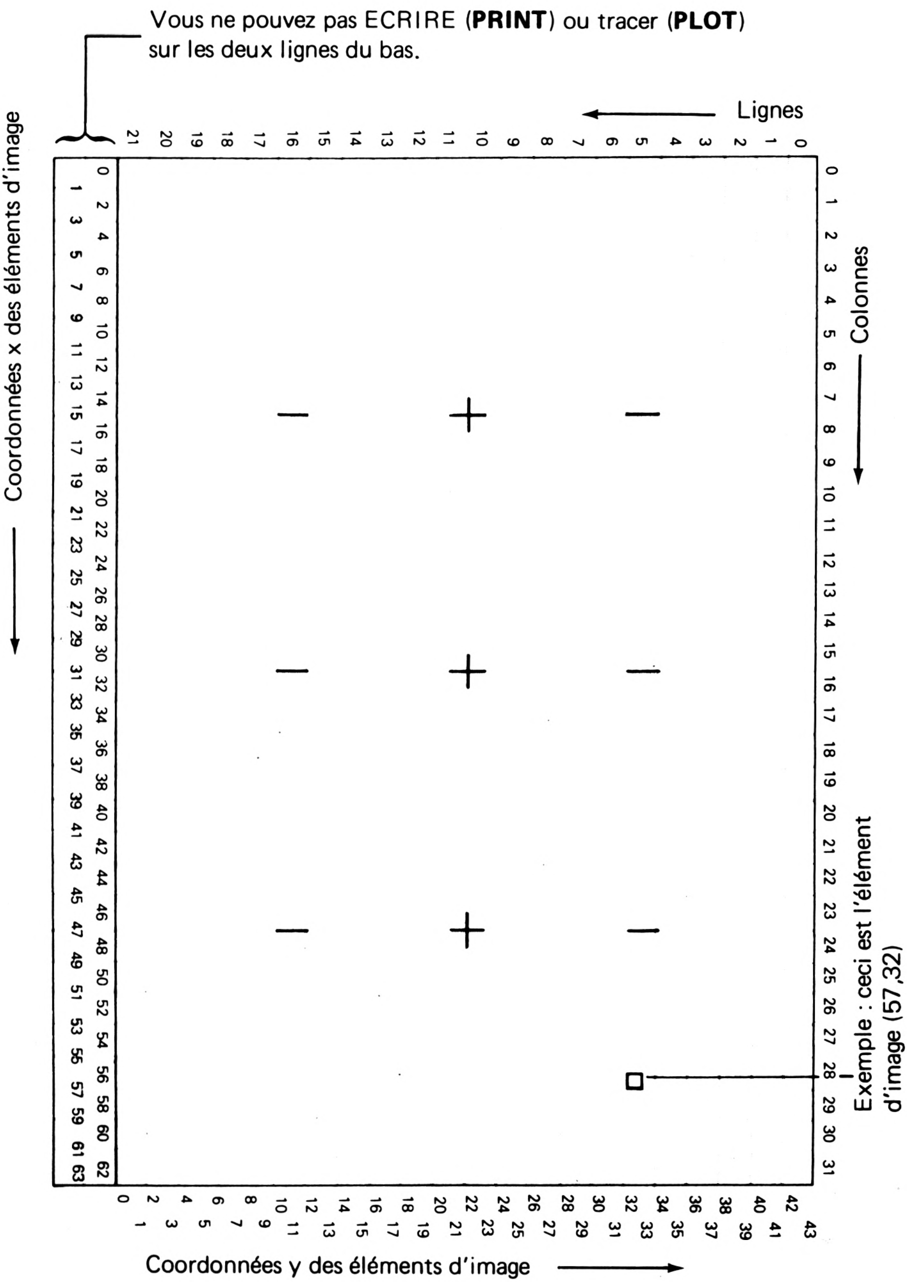
```
1000 LET U=C-A
1005 REM U INDIQUE LE NOMBRE DE PAS A FAIRE
1010 LET V=D-B
1015 REM V INDIQUE LE NOMBRE DE PAS PARCOURU
1020 LET D1X=SGN U
1030 LET D1Y=SGN V
1035 REM (D1X,D1Y) EST UN SEUL PAS DANS LA DIRECTION
     DIAGONALE
1040 LET D2X= SGN U
1050 LET D2Y=0
1055 REM (D2X,D2Y) EST UN SEUL PAS A GAUCHE OU A DROITE
1060 LET M=ABS U
1070 LET N=ABS V
1080 IF M>N THEN GOTO 1130
1090 LET D2X=0
1100 LET D2Y=SGN V
1105 REM MAINTENANT (D2X,D2Y) EST UN SEUL PAS VERS LE
     HAUT OU LE BAS
1110 LET M=ABS V
1120 LET N=ABS U
1130 REM M EST LE PLUS GRAND DE ABSU ET ABSV, N EST LE
     PLUS PETIT
1140 LET S=INT (M/2)
1145 REM NOUS VOULONS PASSER DE (A,B) A (C,D) EN M PAS PAR
     N PAS VERS LE HAUT-BAS OU A DROITE-GAUCHE DE
     GRANDEUR D2, ET PAR M-N PAS DIAGONAUX DE LONGUEUR
     D1, DISTRIBUES AUSSI REGULIEREMENT QUE POSSIBLE
1150 FOR I=0 TO M
1160 PLOT A,B
1170 LET S=S+N
1180 IF S<M THEN GOTO 1230
1190 LET S=S-M
1200 LET A=A+D1X
1210 LET B=B+D1Y
1215 REM PAS DIAGONAL
1220 GOTO 1250
1230 LET A=A+D2X
1240 LET B=B+D2Y
1245 REM PAS VERS LE HAUT-BAS OU A DROITE-GAUCHE
1250 NEXT I
1260 RETURN
```

La dernière partie (lignes 115Ø et suivantes) combine de façon égale M-N mouvements D1 avec N mouvements D2. Imaginez un jeu Monopoly, dont le pourtour comporte M carrés, numérotés de Ø à M-1. Le carré où vous vous trouvez à chaque moment, est numéroté S, en partant du coin opposé au carré GO. Chaque mouvement vous déplace de N carrés autour du jeu, et dans la ligne droite de l'écran, vous faites un mouvement de gauche à droite, ou de haut en bas (en passant par GO), sinon vous faites un mouvement en diagonale. Puisque vos déplacements sur le tableau vous font passer M*N carrés, vous accomplissez N tours complets, vous passez N fois le carré GO; et dans le total de mouvements M il y en a N du type gauche-droite/ de haut en bas, répartis de façon égale.



Ajustez votre programme de telle façon que, si un autre paramètre, E = 1, la ligne est tracée en blanc (employez **UNPLOT**).

Ainsi vous pouvez effacer une ligne que vous venez de tracer, en la retraçant.


Vous ne pouvez pas ECRIRE (**PRINT**) ou tracer (**PLOT**) sur les deux lignes du bas.
Lignes
Colonnes
Coordonnées x des éléments d'image
Coordonnées y des éléments d'image
Exemple : ceci est l'élément d'image (57,32)

TEMPS ET MOUVEMENT

Dans la plupart des cas, vous aimeriez faire un programme d'une longueur spécifique. A cette fin, vous verrez que la déclaration **PAUSE** est très utile (surtout quand vous employez la grande vitesse).

PAUSE n

arrête l'ordinateur et montre n images (à 50 images par seconde, et 60 en Amérique). n peut être égal à 32767, ce qui donne un peu moins que 11 min., quand n a une valeur supérieure à 32767, vous avez une **PAUSE** pour toujours.

Il est toujours possible d'arrêter une pause en enfonçant n'importe quelle touche (retenez qu'un espace ou £ résulte également dans une pause). Vous devez enfoncer la touche après avoir activé la pause. A la fin de la pause, l'écran clignotera. Quand vous employez une déclaration **PAUSE** dans un programme exécuté à vitesse rapide ou avec le ZX80 avec le nouveau K8 **ROM**, la déclaration **PAUSE** doit être suivie de **POKE** 16437,255. Vous verrez que, même si vous ne le faites pas, la pause s'instaure, mais tout le programme risque d'être effacé.

Ce programme-ci fait marcher la trotteuse d'une montre (ici il ne s'agit pas d'une aiguille mais d'un petit point au bord de la montre).

```
  5 REM DESSINONS L'AVANT DE L'HORLOGE
 10 FOR N=1 TO 12
 20 PRINT AT 10-10*COS (N/6*PI),10+10*SIN (N/6*PI);N
 30 NEXT N
 35 REM DEMARRONS L'HORLOGE
 40 FOR T=0 TO 10000
 45 REM T EST LE TEMPS EN SECONDES
 50 LET A=T/30*PI
 60 LET SX=21+18*SIN A
 70 LET SY=22+18*COS A
200 PLOT SX,SY
300 PAUSE 42
310 POKE 16437,255
320 UNPLOT SX,SY
400 NEXT T
```

(laissez tomber les déclarations **REM** quand vous ne disposez pas d'un tableau d'extension de mémoire). Cette montre s'arrêtera après environ 2 3/4 heures à cause de la ligne 40, mais il n'est pas difficile d'allonger ce programme. Notez comment le timing est contrôlé par la ligne 300. Vous pourriez vous attendre à ce que **PAUSE 50** fasse marcher l'aiguille (le point) une fois par seconde, mais le calcul prend également du temps, et l'ordinateur doit nécessairement avoir ce temps.

Ce timing peut être contrôlé avec la méthode 'trail and error' (la méthode expérimentale) en comparant la montre-ordinateur avec une vraie montre et en ajustant la ligne **300** jusqu'à ce qu'ils soient identiques (il est impossible de le faire précisément, le réglage d'une image par seconde correspond à 2% ou une demi-heure par jour).

La fonction **INKEY$** (qui n'a pas d'argument), parcourt le clavier. Quand vous n'enfoncez qu'une seule touche (ou **SHIFT** et une autre), le résultat est le caractère donné par cette touche dans le mode **L**, sinon le résultat est la chaîne vide. Les caractères de contrôle n'ont pas leur effet normal, mais résultent en **CHR$** 118 pour **NEWLINE** – ils sont imprimés comme "?".

Essayez ce programme qui fonctionne comme une machine à écrire:

```
10 IF INKEY$<>"" THEN GOTO 10
20 IF INKEY$="" THEN GOTO 20
30 PRINT INKEY$;
40 GOTO 10
```

Dans cet exemple, la ligne **10** attend jusqu'à ce que vous relachiez le clavier, et la ligne **20** attend jusqu'à ce que vous enfonciez une nouvelle touche.

Sachez que, contrairement à la déclaration **INPUT**, la déclaration **INKEY$** ne vous attend pas. Ainsi vous n'imprimez pas **NEWLINE** mais, par contre, si vous n'imprimez rien du tout, vous avez manqué votre chance.

Exercices:

1. Qu'est-ce qui se passe si vous oubliez la ligne **10** dans le programme machine à écrire?

2. Pourquoi est-il impossible d'écrire un espace ou £ dans ce programme? L'exemple suivant est un programme modifié qui vous donne la possibilité d'écrire des espaces en employant la touche "cursor right" (= **SHIFT** + 8).

```
 10 IF INKEY$<>"" THEN GOTO 10
 20 IF INKEY$="" THEN GOTO 20
 30 LET A$=INKEY$
 40 IF A$=CHR$ 115 THEN GOTO 110
 90 PRINT A$;
100 GOTO 10
110 PRINT " ";
120 GOTO 10
```

Sachez que vous devez lire la ligne **30** comme: "**INKEY$** INTO A$ (**INKEY$** dans A$). Il serait possible d'omettre cette ligne et de remplacer A$ par **INKEY$** dans les lignes **40** et **90**, mais la possibilité existe que les **INKEY$** changent entre les lignes.

Allongez encore un peu ce programme. Quand vous écrivez **NEWLINE** (**CHR$** 118), vous recevez une nouvelle ligne.

3. Une autre façon d'employer les **INKEY$** est de joindre à **PAUSE**, comme dans ce programme alternatif machine à écrire.

```
10 PAUSE 40000
20 POKE 16437,255
30 PRINT INKEY$;
40 GOTO 10
```

Pourquoi est-ce essentiel au bon fonctionnement que la **PAUSE** ne finisse pas au moment où vous pressez une autre touche ? Cette méthode a l'inconvénient que l'écran clignote, mais c'est la seule façon d'agir quand on emploie le mode rapide. Exécutez le programme en mode rapide et remarquez que l'ordinateur profite de la pause pour montrer l'image du téléviseur.

4. Ce programme-ci vous rendra sûrement fou. L'ordinateur vous montre un chiffre que vous (ou une autre victime innocente) lui renvoyez par écrit. Au début vous avez une seconde. Si vous ne réussissez pas, vous aurez un laps de temps plus long pour le chiffre prochain. Si, par contre, vous réussissez la première fois, le laps de temps sera plus court après. Le but est d'agir le plus vite possible. Enfoncez maintenant la touche Q et attendez le résultat, plus votre score est élevé mieux ça vaut.

```
10 LET T=50
15 REM T=NOMBRE DE TRAMES PAR GO—INITIALLEMENT 5Ø
   PAR SECONDE
20 SCROLL
30 LET A$=CHR$ INT (RND*10+CODE "0")
35 REM A$ EST UN NOMBRE ALEATOIRE
40 PRINT A$
45 PAUSE T
50 POKE 16437,255
60 LET B$=INKEY$
70 IF B$="Q" THEN GOTO 200
80 IF A$=B$ THEN GOTO 150
90 PRINT "RATE"
100 LET T=T*1.1
110 GOTO 20
150 PRINT "OK"
160 LET T=T*0.9
170 GOTO 20
200 SCROLL
210 PRINT "VOTRE SCORE EST"; INT (5ØØ/T)
```

5. (Seulement pour ceux qui disposent d'un **RAM** supplémentaire (Random Acces Memory).

Employez la routine rectiligne du chapitre 15; changez le programme de la trotteuse pour qu'il montre également les aiguilles des minutes et des heures, en les marquant chaque minute. (Prenez une aiguille plus courte pour les heures). Si vous êtes ambitieux, modifiez le programme de telle façon qu'il fasse une petite exhibition tous les quarts d'heure.

6. Pour le plaisir, essayez ceci:

```
10 IF INKEY$ = "" THEN GOTO 10
20 PRINT AT 11,14; "AIE"
30 IF INKEY$<>"" THEN GOTO 30
40 PRINT AT 11,14; "   "
50 GOTO 10
```

L'IMPRIMANTE DU ZX81

Si vous possédez l'imprimante du ZX81, vous trouverez également les instructions de commande. Ce chapitre comprend les déclarations BASIC qui font fonctionner l'ordinateur. Les deux premiers, **LLIST** et **LPRINT**, sont comme **LIST** et **PRINT**, excepté qu'ils opèrent sur l'imprimante et non pas sur le téléviseur. L est le témoin d'un accident historique.

Quand le BASIC a été inventé, on se servait surtout d'une simple machine à écrire électrique au lieu d'un téléviseur, de sorte que **PRINT** voulait vraiment dire "imprimer". Si on avait énormément de données à imprimer, on employait une imprimante qui fonctionnait ligne par ligne et très vite. **LPRINT** signifiait: imprimer ligne par ligne.
Essayez ce programme-ci:

```
10 LPRINT "CE PROGRAMME:"...
20 LLIST
30 LPRINT,,"ECRIT LE JEU DE CARACTERES" ...
40 FOR N=0 TO 255
50 LPRINT CHR$ N;
60 NEXT N
```

La troisième déclaration, **COPY**, imprime une vraie copie de l'écran du téléviseur. Par ex.: mettez la liste du programme précédent en image et frappez:

COPY

Vous pouvez toujours arrêter l'imprimante en enfonçant la touche **BREAK** (espace). Si vous le faites sans qu'une imprimante soit connectée à l'ordinateur, vous perdez tous les éléments **OUTPUT** et l'ordinateur continue avec la prochaine déclaration. De temps en temps, l'ordinateur est bloqué. Dans ce cas, utilisez la touche **BREAK** pour le libérer.

Sommaire:
Déclarations: **LPRINT**, **LLIST** et **COPY**.

Remarque:
Aucune de ces déclarations ne s'emploie dans le BASIC courant. Néanmoins, **LPRINT** est employé sur d'autres ordinateurs.

Exercices:
1. Essayez ceci:

```
10 FOR N = 31 TO 0 STEP -1
20 PRINT AT 31 - N, N; CHR$ (CODE "0" + N)
30 NEXT N
```

Vous remarquerez une structure de lettres qui descendent du coin en haut à gauche jusqu'en bas de l'écran. Le programme s'arrête avec le rapport d'erreur "5". Changez '**AT** 31 - 1, N' de la ligne 2Ø en '**TAB** N'. Ce programme aura le même effet qu'auparavant.

Changez **PRINT** de la ligne 2Ø en **LPRINT**. Cette fois-ci, il n'aura plus de rapport d'erreur 5, puisque cela n'existe pas quand on travaille avec l'imprimante et la structure continuera pour 1Ø lignes supplémentaires.

Changez '**TAB** N' en '**AT** 21 - N,N' en utilisant **LPRINT**. Cette fois-ci vous aurez juste une ligne de symboles. La raison de cette différence est que l'**OUTPUT** de **LPRINT** n'est pas immédiatement imprimé, mais gardé dans une mémoire tampon, sur une ligne que l'ordinateur imprimera dès qu'il a terminé.

L'imprimante commence quand:

(i) la mémoire tampon est pleine;
(ii) après une déclaration **PRINT** qui n'est pas suivie d'une virgule ou d'un point-virgule;
(iii) quand une virgule ou un élément **TAB** exige une nouvelle ligne;
(iv) à la fin d'un programme, quand il reste encore quelque chose à imprimer.

(iii) Explique le fonctionnement du programme avec **TAB**. Comme pour **AT**, on ignore le numéro de ligne et la position **LPRINT** est changé en un numéro de colonne (comme pour la position **PRINT**, mais pour l'imprimante à la place du téléviseur). Un élément **AT** ne peut jamais envoyer une ligne à l'imprimante. En fait, le numéro de ligne après **AT** n'est pas complètement ignoré. Il faut un numéro entre -21 et +21 pour éviter une erreur. C'est la raison pour laquelle il vaut mieux spécifier la ligne Ø. L'élément '**AT** 21 - N,N' de la version finale de notre programme sera beaucoup plus avantageux, bien que moins illustratif s'il était remplacé par '**AT** Ø, N'.

2. Faites un graphique imprimé de **SIN** en exécutant le programme du chapitre 18 et en employant **COPY**.

LES SOUS-CHAINES

La sous-chaîne d'une chaîne consiste d'un certain nombre de caractères consécutifs, pris dans l'ordre. Ainsi, "STRING" est une sous-chaîne de "BIGGER STRING", mais "B STING" et "BIG REG" ne le sont pas. Il existe une expression pour décrire les sous-chaînes: "sicling" (= trancher) et ceci peut être appliqué à n'importe quelle chaîne. La forme qui est généralement employée est:

string expression (start **TO** finish)

de telle façon que:

"ABCDEF" (2 **TO** 5) = "BCDE"

Si vous oubliez le début, le 1 est également compté; si vous oubliez la fin, la chaîne est considérée dans toute sa longueur.
Ainsi:

"ABCDEF" (**TO** 5) = "ABCDEF" (1 **TO** 5) = "ABCDE"
"ABCDEF" (2 **TO**) = "ABCDEF" (2 **TO** 6) = "BCDEF"

et

"ABCDEF" (**TO**) = "ABCDEF" (1 **TO** 6) = "ABCDEF"

(Vous pouvez également écrire "ABCDEF" ( ). Il existe une forme quelque peu différente qui omet le **TO** et qui ne mentionne qu'un seul chiffre.

"ABCDEF" (3) = "ABCDEF" (3 **TO** 3) = "C"

Normalement le début et la fin doivent se référer à une partie existante de la chaîne, mais il y a une règle qui prime la précédente: si le début est un chiffre plus grand que le chiffre de la fin, le résultat est la chaîne vide, donc:

"ABCDEF" (5 **TO** 7)

donne l'erreur 3 (erreur du dernier indice) parce que la chaîne ne compte que 7 caractères, 7 est donc un chiffre trop grand, mais:

"ABCDEF" (8 **TO** 7) = " "

et

"ABCDEF" (1 **TO** Ø) = " "

Ni le début, ni la fin ne peuvent être des chiffres négatifs, sinon vous recevez l'erreur B.

Le prochain programme rend B$ égal à A$, mais en omettant toutes les espaces.

```
10 INPUT A$
20 FOR N=LEN A$ TO 1 STEP -1
30 IF A$(N)<>" " THEN GOTO 50
40 NEXT N
50 LET B$=A$( TO N)
60 PRINT """";A$;"""","""";B$;""""
70 GOTO 10
```

Remarquez que, si A$ est entièrement vide, on a à la ligne 50: N = 0 & A$ (**TO** N) = A$ (1 **TO** 0) = " ".

En ce qui concerne les variables de chaînes, nous ne pouvons pas seulement en extraire de sous-chaînes, mais nous pouvons également y affecter.
Par ex., écrivez:

LET A$ = "LOR LOVE A DUCK"

et puis:

LET A$ (5 **TO** 8) = "******"

et

PRINT A$

Remarquez que, puisque la sous-chaîne A$ (5 **TO** 8) n'a que 4 caractères, on n'a utilisé que 4 étoiles. C'est une caractéristique d'affectation des sous-chaînes. La sous-chaîne doit avoir exactement la même longueur qu'avant. Afin d'être certain que ce soit ainsi, la chaîne affectée est coupée à droite si elle est trop longue, ou remplie d'espaces si elle est trop courte.

Ceci s'appelle une affectation procrustienne, d'après l'aubergiste Procrust qui allongeait ses clients sur un chevalet et les raccourcissait en leur coupant les pieds pour les ajuster à leur lit.
Essayez maintenant:

LET A$ () = "COR BLIMEY"

et

PRINT A$; "."

Vous verrez que la même chose se passe à nouveau (cette fois-ci avec des espaces) parce que A$ a la valeur d'une sous-chaîne.

LET A$ = "COR BLIMEY"

Cette déclaration fera bien l'affaire.
"Trancher" (sclicing) peut être considéré comme ayant la priorité 12, donc,

LEN "ABCDEF" (2 **TO** 5) = **LEN** ("ABCDEF" (2 **TO** 5)) = 4

Avant de trancher des expressions compliquées de chaînes, il leur faut des parenthèses. Par exemple:

"ABC" + "DEF" (1 **TO** 2) = "ABCDE"
("ABC" + "DEF") (1 **TO** 2) = "AB"

Sommaire:
"Trancher" (sclicing) et l'emploi de **TO**.
Remarquez que cette notation n'est pas standardisée.

Exercices:
1. Certains BASICs (pas le BASIC du ZX81) ont des fonctions dites: **LEFT$**, **RIGHT$**, **MID$** et **TL$**.
**LEFT$** (A$,N) donne une sous-chaîne de A$ comportant les N premiers caractères.
**RIGHT$** (A$,N) donne une sous-chaîne de A$ comportant les caractères commençant par le Nième.
**MIDS$** (A$, N1,N2) donne une sous-chaîne de A$ comportant N2 caractères commençant par le N1ième.
**TL$** (A$) donne une sous-chaîne de A$ comportant tous les caractères sauf le premier.
Comment les écririez-vous dans le BASIC du ZX81?

2. Essayez cette séquence de commandes:

A$ est une chaîne qui contient des parenthèses. Ainsi, vous pouvez toujours imprimer de telles chaînes, si vous êtes persévérant, mais enfin, si vous aviez imprimé, dès le début:

LET A$ = "X" + "Y"

la partie à droite de la marque d'égalité aurait été traitée comme étant une expression avec A$ = "XY".
Ecrivez maintenant:

LET B$ = "X"" + ""Y"

Vous verrez que, quoique A$ et B$ paraissent semblables sur l'écran, ils sont bien différents.
Essayez:

PRINT A$ = B$

Là où B$ ne contient que des caractères de symboles de citation, A$ contient des caractères normaux de parenthèses.

3. Exécutez ce programme-ci:

```
 10 LET A$ = "LEN""ABCD"""
100 PRINT A$; "=" ; VAL A$
```

Ce programme ne fonctionnera pas, puisqu'il ne considère pas le symbole de citation " " comme des parenthèses de chaîne.

Ajoutez quelques lignes de plus entre 10 et 100 afin de remplacer les symboles de citation dans A$ par des parenthèses de chaîne (que vous devez appeler **CHR$** 11) et essayez à nouveau. Modifiez également le programme du chapitre 9, ex. 3 et expérimentez un peu avec ce programme.

4. Ce sous-programme efface toute occurence de la chaîne "CARTHAGO" de A$.

```
1000 FOR N=1 TO LEN A$-7
1020 IF A$(N TO N+7)="CARTHAGO" THEN LET
A$(N TO N+7)="********"
1030 NEXT N
1040 RETURN
```

Ecrivez un programme qui donne de diverses valeurs à A$ (par ex. "DELENDA EST CARTHAGO" et qui contient les sous-programmes.

LES TABLEAUX

Supposons que vous avez une liste de chiffres, par ex. le nombre de collecteurs d'impôts décédés par mois, durant l'année fiscale courante. Pour emmagasiner ces données dans l'ordinateur, vous pourriez proposer une variable par mois, ce qui ne serait pas très pratique. Vous pourriez appeler ces variables: ALAS NO MORE 1, ALAS NO MORE 2, etc... jusqu'à ALAS NO MORE 12, mais le programme qui imprimerait ces douzes chiffres serait long et ennuyeux à introduire.
Il serait bien plus élégant d'écrire:

```
 5 REM CE PROGRAMME NE VA PAS MARCHER
1Ø FOR N=1 TO 12
2Ø PRINT HELAS PLUS DE N
3Ø NEXT N
```

Eh bien, c'est impossible.

Mais il existe un mécanisme qui applique la même idée et qui emploie des tableaux (arrays).

Un tableau est un ensemble de variables ou d'éléments qui ont tous le même nom et qui se différencient d'un chiffre écrit entre parenthèses, après le nom.

Dans notre exemple, le nom sera A (comme pour les variables de contrôle des boucles (loopings) **FOR-NEXT**, le nom ne peut comporter qu'une seule lettre), et les douze variables seront A(1), A(2),... A(12).

Les éléments d'un tableau s'appellent des variables indicées, afin de ne pas les confondre avec les variables que vous connaissez déjà.

Avant d'employer un tableau, il faut lui réserver une place dans l'ordinateur, et à cette fin, vous utilisez une déclaration **DIM** (dimension).

```
DIM A(12)
```

crée un tableau du nom de A avec la dimension 12 (il y a douze variables indicées: A(1), A(2),... A(12)) et fait initier les 12 valeurs par Ø. De surcroit, il détruit un tableau A qui existerait déjà. (Mais pas une simple variable. Un tableau et une simple variable du même nom peuvent co-exister, et il est impossible de les confondre puisqu'un tableau est toujours indicé).

L'indice peut être n'importe quelle expression numérique, donc vous pouvez écrire:

```
1Ø FOR N = 1 TO 12
2Ø PRINT A(N)
3Ø NEXT N
```

Vous pouvez également créer des tableaux de plusieurs dimensions. Dans un tableau à deux dimensions, vous devez avoir deux chiffres pour spécifier les éléments – comme les numéros de lignes et de colonnes qui définissent une position sur l'écran – donc cela aura la forme d'une table.

Vous pouvez également définir des numéros de lignes et de colonnes, pour référer à une page imprimée. Prévoyez encore une dimension supplémentaire pour indiquer les pages.

Evidemment, nous parlons ici de tableaux numériques. Les éléments ne sont pas des caractères imprimés comme dans un livre, mais des numéros. Imaginez-vous un tableau à trois dimensions C qui est spécifié comme C (numéro de page, numéro de ligne, numéro de colonne).

Pour faire un tableau de deux dimensions B, avec les dimensions 3 et 6, vous utilisez une déclaration **DIM**

DIM (3,6)

ce qui vous donne 3*6 = 18 variables indicées:

B(1,1), B(1,2), . . . , B(1,6)
B(2,1), B(2,2), . . . , B(2,6)
B(3,1), B(3,2), . . . , B(3,6)

Le même principe vaut pour n'importe quel nombre de dimensions.

Il existe également des chaînes de tableaux. Elles se différencient des chaînes normales parce qu'elles ont une longueur fixe et vous pouvez toujours les affecter à la façon de Procrust. Une autre manière de les considérer est comme étant des tableaux (avec une dimension supplémentaire) de caractères simples.

Le nom d'une chaîne de tableau se compose d'une lettre suivie de $. Une chaîne de tableau et un tableau simple ne peuvent pas avoir le même nom (Cfr. les numéros).

Supposez que vous avez un tableau de 5 chaînes: A$. Maintenant vous devez décider sur la longueur de ces chaînes – supposons que 1Ø caractères par chaîne suffisent. Ecrivez:

DIM A$ (5,1Ø)

Ainsi, vous créez un tableau 5* 1Ø de caractères, mais vous pouvez également considérer chaque rangée comme étant une chaîne.

A$(1) =	A$(1,1)	A$(1,2)	...	A$(1,1Ø)
A$(2) =	A$(2,1)	A$(2,2)	...	A$(2,1Ø)
:	:	:	:	:
A$(5) =	A$(5,1)	A$(5,2)	...	A$(5,1Ø)

Si vous donnez le même nombre d'indice que vous aviez de dimensions dans la déclaration **DIM**, vous aurez un seul caractère. Si vous omettez le dernier, vous aurez une chaîne de longueur fixe. Par ex.: A$ (2,7) est le 7ième caractère de la chaîne A$ (2). En employant la notation tranchée (sliced notation), nous pourrions écrire A$ (2),(7).

Ecrivez maintenant:

LET A$ (2) = "1234567890"

et

PRINT A$ (2)/ A$ (2,7)

le résultat sera:

1234567890 7

Vous pouvez également employer une tranche à la place du dernier indice (celui que vous pouvez omettre). Par exemple:

A$(2,4 **TO** 8) = A$ (2) (4 **TO** 8) = "45678"

Retenez: Dans une chaîne de tableaux, toutes les chaînes ont la même longueur. La déclaration **DIM** a un numéro supplémentaire pour spécifier sa longueur. Quand vous écrivez une variable indicée d'une chaîne de tableaux, vous pouvez y ajouter un numéro supplémentaire qui correspond au numéro supplémentaire de la déclaration **DIM**.

Sommaire:
Les tableaux.
La déclaration **DIM.**

Exercices:

1. Créez un tableau M$ de douze chaînes, dans lequel M$(N) est le nom du Nième mois. (Un conseil: la déclaration **DIM** sera écrite comme **DIM** M$ (12,9).

Contrôlez-le en imprimant toutes les M$ (N). (employez des boucles).
Ecrivez:

PRINT "NOW IS THE MONTH OF"; M$(5); "ING";
"WHEN MERRY LADS ARE PLAYING"

Que pourriez-vous faire de toutes ces espaces?

2. Il existe des chaînes de tableaux sans dimension. Ecrivez:

DIM A$ (10)

et vous verrez que A$ se comporte comme une variable de chaîne, mais que la longueur sera toujours 10 et que la manière d'affecter de Procrust pourra toujours être appliquée.

3. **READ, DATA** et **RESTORE**: qui est-ce qui en a besoin?

La plupart des BASICs (mais pas celui du ZX81) contiennent 3 déclarations, soit **READ, DATA** et **RESTORE.**

La déclaration **DATA** est une liste d'expressions. Quand on prend toutes les déclarations **DATA** d'un programme, on reçoit toute une liste d'expressions: la liste **DATA.**

La déclaration **READ** est employée pour affecter des expressions à des variables.

READ X

par exemple affecte l'expression en cours de la liste **DATA** à la variable X et continue vers la prochaine expression pour la prochaine déclaration **READ**.

RESTORE repart du début de la liste **DATA**.

En théorie, vous pouvez toujours remplacer la déclaration **READ** et **DATA** par des déclarations **LET**, bien que leur fonction principale soit de commencer un tableau, comme dans le programme suivant:

```
5  REM CE PROGRAMME NE PEUT PAS FONCTIONNER EN BASIC
   ZX81
1Ø DIM M$(12,3)
2Ø FOR N=1 TO 12
3Ø READ M$(N)
4Ø NEXT N
5Ø DATA "JAN","FEV","MAR","AVR"
6Ø DATA "MAI","JUN","JUI","AOU"
7Ø DATA "SEP","OCT","NOV","DEC"
```

Si vous ne désirez exécuter ce programme qu'une seule fois, vous pouvez remplacer la ligne 3Ø par une déclaration **INPUT**. Ainsi:

```
1Ø DIM M$(12,3)
2Ø FOR N=1 TO 12
3Ø INPUT M$(N)
4Ø NEXT N
```

et vous n'aurez pas à écrire davantage.

Néanmoins, si vous voulez garder le programme, vous n'allez certainement pas réécrire tous les mois, chaque fois que vous en aurez besoin.

Employez la méthode suivante:

(i) Commencez le tableau avec le programme employé ci-dessous.

(ii) Concevez le programme initial (N'employez pas **NEW**, parce que vous voulez garder votre tableau par après).

(iii) Ecrivez le reste du programme et gardez-le. Ainsi, les variables, aussi bien que le tableau entier seront gardées.

(iv) Quand vous rechargez le programme, rechargez également le tableau.

(v) N'employez pas **RUN** pour exécuter le programme, parce qu'il supprime les variables. Employez plutôt **GOTO** avec le numéro de ligne.
Vous pouvez également employer la technique **LOAD** et exécution du chapitre 16, exercice 3. Vous devez alors employer la déclaration **SAVE** dans le point (iii) ci-dessus et le point (v) peut être omis.

QUAND L'ORDINATEUR EST SATURE

Le ZX81 a une quantité limitée de stockage interne, et ce n'est pas difficile de la remplir. Au moment où la mémoire est remplie, vous êtes averti par un rapport d'erreur 4, mais il est possible qu'il se passe autre chose, et ce sont souvent des choses assez bizarres.

La réaction dépend du fait que vous disposez d'un dispositif d'extension de mémoire ou pas.

Imaginez que vous n'en avez pas, et si vous en avez un, débranchez-le (après avoir éteint l'ordinateur).

Le fichier d'affichage, qui est l'endroit dans l'ordinateur où sont emmagasinées toutes les images télévisées, est conçu de telle sorte qu'il ne prend que l'espace qui a déjà été employé.

Une ligne de l'écran consiste en 32 caractères et un caractère **NEWLINE**. Cela veut dire que, en imprimant, vous pouvez dépasser la capacité de la mémoire. Ceci se passe souvent en faisant des listes.
Imprimez:

NEW
DIM A (150)
1Ø **FOR** N = 1 **TO** 15
2Ø **PRINT** N

Voici la première surprise: la ligne 1Ø disparaît de la liste. La liste doit contenir, à tout prix, la ligne courante 2Ø, et il n'y a pas de place pour les deux.
Ecrivez maintenant:

3Ø **NEXT** N

Il n'y a, à nouveau, que de la place pour la ligne 3Ø. Ecrivez maintenant:

4Ø **REM** X (sans **NEWLINE**)

et vous verrez comment la ligne 3Ø disparaît et comment la ligne 4Ø se déplace vers le haut de l'écran. Elle n'est pas entrée dans le programme – vous disposez toujours du curseur **L** qui peut se déplacer. Tout ce que vous avez vu est un mécanisme qui donne 24 lignes à la partie la plus basse de l'écran afin de donner la priorité sur la partie la plus haute.
Ecrivez maintenant:

XXXXXX (sans **NEWLINE**)

et le curseur disparaîtra – il n'y a pas de place pour l'imprimer. Ecrivez un X sans **NEWLINE** et un des X disparaîtra. Ecrivez ensuite **NEWLINE**.

Tout disparaîtra, mais le programme reste dans l'ordinateur, ce que vous pouvez prouver en omettant la ligne 1Ø et en employant et

Ecrivez maintenant:

10 **FOR** N = 1 **TO** 15

et le tout se déplacera de nouveau vers le haut de l'écran, comme la ligne 40. Mais quand vous enfoncez **NEWLINE**, le message n'entrera pas dans la mémoire, quoiqu'il n'y ait pas de rapport d'erreur ou de marqueur **S** pour indiquer qu'il y a quelque chose qui ne va pas. C'est parce qu'il n'y a pas de place pour contrôler la syntaxe d'une ligne, mais normalement cela se passe seulement avec des lignes contenant des chiffres (autres que ceux avec lesquels commencent les lignes).

La seule façon d'y remédier, c'est de faire de la place, mais faites d'abord disparaître la ligne 10 qui n'entre tout de même pas.

Enfoncez **EDIT**: l'écran sera effacé car il n'y a pas de place pour laisser descendre la ligne. Quand **EDIT** n'efface rien, vous pouvez toujours essayer d'imprimer un certain nombre d'espaces jusqu'à ce que le curseur monte sur l'écran. Enfoncez **NEWLINE** et vous verrez qu'une partie de la liste revient. Omettez maintenant la ligne 40 (dont vous n'aviez tout de même pas besoin) en écrivant:

40 (et **NEWLINE**)

Essayez maintenant d'imprimer la ligne 10 – ça n'ira pas encore. Effacez-la à nouveau. Vous devez encore trouver un peu d'espace supplémentaire. Rappelez-vous la raison pour laquelle la ligne 10 était rejetée, c'était probablement puisqu'il n'y avait pas de place pour contrôler la syntaxe des deux chiffres, 1 et 15. Si vous omettez la ligne 20 du programme, vous aurez peut-être de la place pour placer la ligne 10, et pour replacer la ligne 20 (qui ne contient pas de chiffres) par après.
Essayez ceci: écrivez:

20
10 **FOR** N = 1 **TO** 5
20 **PRINT** N

et le programme entre facilement.
Ecrivez:

GOTO 10

et vous verrez que cette ligne est de nouveau rejetée, parce que la syntaxe ne peut pas être contrôlée. Mais quand vous l'effacez et vous écrivez:

RUN

tout est en ordre (**RUN** balaie les environs et fait beaucoup de place). Réécrivez maintenant tout le programme en commençant par **NEW** jusqu'à la ligne 30 et puis:

40 **REM** XXXXXXXXXXXX

(12 Xs), ce qui finira par ressembler 4Ø RE. Quand vous enfoncez **NEWLINE**, la liste ne comprendra que la ligne 3Ø et en fait, la ligne 4Ø sera totalement perdue. C'était parce que simplement trop long pour s'ajuster dans le programme. Le résultat est encore plus mauvais quand il s'agit d'une ligne qui est la version allongée d'une ligne qui est déjà dans le programme. Vous perdez aussi bien l'ancienne ligne que la nouvelle (ligne de remplacement). Afin d'y remédier, il faut se procurer un paquet RAM (RANDOM ACCESS MEMORY) qui s'ajuste au dos de l'ordinateur. Le paquet RAM 16 K de Sinclair donne à l'ordinateur une mémoire 16 fois plus importante. Un ancien paquet RAM 3 K du ZX80 ne s'ajuste pas au ZX81.

L'ordinateur réagit quelque peu différemment avec le RAM, parce que le fichier de l'écran est rempli d'espaces pour former des lignes de 32 caractères (remarquez que **SCROLL** bouleverse cette situation, Cfr. chapitre 27). Maintenant, quand vous imprimez et enregistrez, la mémoire sera suffisamment grande et les listes ne seront pas raccourcies, mais vous verrez que les lignes seront toujours entassées ou perdues. Une fois de plus, la seule solution est de trouver un peu d'espace quelque part. Si vous disposez d'une extension de mémoire, installez-la et imprimez tout ce qu'on a vu dans ce chapitre en employant:

DIM A (3Ø69)

pour remplacer:

DIM A (150)

C'est assez compliqué de résumer et la morale est: évitez d'embrouiller l'ordinateur. Néanmoins, la morale suivante est que les choses ne sont pas toujours aussi compliquées qu'elles en ont l'air.

1. Quand les listes commencent à raccourcir et que tout commence à sauter d'une place à l'autre, cela veut dire que l'espace devient un peu juste.

2. Quand **NEWLINE** ne semble plus avoir de l'effet à la fin d'une ligne, il n'y a probablement plus de place pour un autre chiffre. Effacez la ligne en employant **EDIT**, **NEWLINE** ou **RUBOUT**.

3. **NEWLINE** pourrait perdre une ligne. Pour toutes ces choses bizarres, il existe une seule solution. Pas de panique, cherchez tout de suite de l'espace vide. La première chose à faire: **CLEAR**. Quand vous avez plusieurs variables, et cela ne vous faites rien d'en perdre quelques-unes, effacez-en quelques-unes. Si vous n'en trouvez pas, cherchez des déclarations inutiles dans le programme, comme par exemple une déclaration **REM** et omettez-la.

Sommaire:
Quand la mémoire est remplie, il peut se passer des choses bizarres; mais en général, elles ne sont pas fatales.

COMPTER SUR VOS DOIGTS

Le prochain chapitre creuse un peu l'ordinateur, mais avant de nous y aventurer, nous décrirons comment un ordinateur compte. Il emploie le système binaire, ce qui veut dire qu'il n'a pas de doigts, seulement des pouces.

La plupart des langues européennes comptent en structures plus ou moins régulières. Les Anglais par exemple, quoiqu'ils commencent assez bizarrement, cette façon de compter se régularise assez vite.

vingt, vingt-et-un, vingt-deux, ..., vingt-neuf
trente, trente-et-un, trente-deux, ..., trente-neuf
quarante, quarante-et-un, quarante-deux, ..., quarante-neuf

etc... et ceci devient encore plus systématique avec les chiffres arabes que nous employons. La seule raison d'employer des dizaines est que nous possédons 10 doigts. Supposons que les Martiens ont 3 doigts supplémentaires à chaque main (si on peut appeler cela des doigts). Ils emploient un système hexadécimal (ou hex), basé sur le chiffre 16, au lieu de notre système décimal. Ils ont besoin de 6 chiffres hex supplémentaires et ils les écrivent comme A, B, C, D, E, F. Et qu'est-ce qu'il y a après le F? Comme nous avons nos dix doigts, nous écrivons 10 pour dix, eux écrivent 10 pour le chiffre seize.

Leur système numérique commence comme le nôtre,

Hex	*Français*
Ø	Zéro
1	Un
2	Deux
:	:
:	:
9	Neuf

mais ensuite il continue comme suit:

A	Dix
B	Onze
C	Douze
D	Treize
E	Quatorze
F	Quinze
10	Seize
11	Dix-sept
:	:
:	:
19	Vingt-cinq
1A	Vingt-six

Hex	*Français*
1B	Vingt-sept
:	:
:	:
1F	Trente-et-un
20	Trente-deux
21	Trente-trois
:	:
:	:
9E	Cent cinquante-huit
9F	Cent cinquante-neuf
AØ	Cent soixante
A1	Cent soixante-et-un
:	:
:	:
FE	Deux cent cinquante-quatre
FF	Deux cent cinquante-cinq
1ØØ	Deux cent cinquante-six

Si vous utilisez la notion HEX et vous désirez en faire une habitude, écrivez 'h' à la fin des chiffres et prononcez HEX. Par exemple: pour 'one hundred and fifty eight' (158), écrivez '8 E h' et prononcez: 'nine E hex'.
Vous vous demandez peut-être ce que tout cela a à faire avec les ordinateurs. En fait, les ordinateurs se comportent comme s'ils n'avaient que 2 chiffres, représentés par un haut voltage, ou "off" (Ø), ou par un bas voltage ou "on" (1).
Ce système s'appelle le système binaire et les deux chiffres binaires s'appellent des BIT, un BIT est donc un Ø ou un 1.
Dans les différents systèmes, le calcul commence comme suit:

Français	*Décimal*	*Hexadécimal*	*Binaire*
Zéro	0	0	0 or 0000
Un	1	1	1 or 0001
Deux	2	2	10 or 0010
Trois	3	3	11 or 0011
Quatre	4	4	100 or 0100
Cinq	5	5	101 or 0101
Six	6	6	110 or 0110
Sept	7	7	111 or 0111
Huit	8	8	1000
Neuf	9	9	1001
Dix	10	A	1010
Onze	11	B	1011
Douze	12	C	1100
Treize	13	D	1101
Quatorze	14	E	1110
Quinze	15	F	1111
Seize	16	10	10000

Le point important est que 16 est égal à 2 à la quatrième puissance, ce qui rend facile la conversion de HEX au système binaire. Pour convertir le système HEX en système binaire, il faut changer chaque chiffre HEX en quatre bits, selon la table donnée plus haut. Pour convertir le système binaire en système HEX, il faut diviser les chiffres binaires en groupes de quatre bits, en partant de la droite et ensuite il faut remplacer les groupes par un chiffre HEX correspondant. C'est pourquoi les hommes notent les chiffres emmagasinés dans l'ordinateur en langage HEX, bien que l'ordinateur emploie le langage binaire. Les BITs à l'intérieur de l'ordinateur sont la plupart du temps groupés en unités de 8 BITs, ce qu'on appelle des BYTES. Un byte peut représenter un chiffre de zéro à 255, 11111111 binaire ou FF hex, ou n'importe quel caractère que le ZX81 possède dans son ensemble de caractères. Sa valeur peut être écrite en deux chiffres hex.

Deux BYTES peuvent être rassemblés pour former un WORD. Un WORD se compose de 16 bits ou quatre valeurs HEX et représente un chiffre de Ø à $2^{16-1} = 65535$. Un byte est toujours égal à 8 bits, mais les words diffèrent d'ordinateur à ordinateur.

Sommaire:
Systèmes décimal, hexadécimal et binaire.
Bits, Bytes et Words.

Exercices

1. L'unité courante employée par les Martiens est la livre (pound) et elle est divisée en 16 onces (ounces). Comment convertir les "ponds et ounces" en ounces et vice versa?
(i) quand tous les chiffres sont écrits en système décimal,
(ii) quand tous les chiffre sont écrits en système hex.

2. Comment convertir du système décimal en système hex? (Cfr. ex. 1)
Ecrivez un programme qui convertit les valeurs numériques en valeurs de chaînes en donnant l'équivalent hex, et vice versa. (C'est ce que font les déclarations **STR$** et **VAL** avec une représentation décimale).

3. Supposons que les habitants de Vénus aient 8 doigts en total (sans pouces). Comment l'ordinateur pourrait-il être utile pour leur système octale (base 8)?

LE FONCTIONNEMENT DE L'ORDINATEUR

L'illustration de la page suivante vous montre l'intérieur de l'ordinateur. Comme vous l'avez remarqué, tout est écrit en abréviations de trois lettres (TLA).

Les pièces en plastique avec les jambes métalliques sont les formidables chips en silicone, auxquels vous devez les prestigieuses montres digitales et le ZX81 de Sinclair. A l'intérieur de chaque pièce en plastique, il y a une pièce en silicone de la grandeur du curseur **K**, connectée à l'aide de fils aux jambes métalliques. La matière grise qui contrôle l'opération est l'unité centrale (Processer Chip, CPU = Central Processer Unit). Ceci est une unité Z80 (en fait une Z80 A qui fonctionne plus vite). Cette unité contrôle tout l'ordinateur, s'occupe de l'arithmétique, tient compte de la touche que vous enfoncez, prend la décision en ce qui concerne le résultat et fait la coordination de l'image télévisée. Malgré son intelligence, il ne serait pas capable de la faire tout seul. Sans aide il ne comprend rien du BASIC, ni de l'arithmétique de la virgule flottante ou du téléviseur. Il cherche son information chez un autre chip, le ROM (READ ONLY MEMORIE). Le ROM n'est rien d'autre qu'une longue liste d'instructions qui fait un programme d'ordinateur et qui ordonne l'unité ce qu'elle doit faire en toute circonstance prévisible. Ce programme n'est pas écrit en BASIC, mais en code machine du ZX80 et prend la forme d'une longue séquence de bytes (un byte est un chiffre entre Ø et 255). Chaque byte a son adresse qui indique approximativement où il se situe dans le ROM. Le premier a l'adresse Ø, le second a l'adresse 1 etc... jusqu'à 8191. Dans l'ensemble cela fait 8192 = 8*1Ø24 bytes. C'est pourquoi le BASIC du ZX81 est appelé un BASIC de 8K. 1K = 1Ø24 ou $2^{1Ø}$.

Bien qu'il y ait des chips semblables pour différentes machines, cette instruction particulière est propre au ZX81 et a été écrite spécialement pour lui par une petite firme de mathématiciens de Cambridge. Vous pouvez contrôler quel byte se situe à quelle adresse en utilisant la fonction **PEEK**. Par ex. ce programme imprime les 21 premiers bytes du ROM (et leurs adresses)

```
1Ø PRINT "ADRESSE";TAB 8; "BYTE"
2Ø FOR A = Ø TO 2Ø
3Ø PRINT A; TAB 8; PEEK A
4Ø NEXT A
```

Le chip suivant à considérer est la mémoire ou le chip RAM (Random Access Memory). C'est l'espace où l'ordinateur emmagasine toute l'information qu'il veut garder, le programme BASIC, les varaibles, les images télévisées et les notes variables (les variables de système) sur l'état de l'ordinateur.

Comme le ROM, la mémoire consiste en bytes, qui possèdent tous une adresse: les adresses vont de 16384 jusqu'à 17407 (ou parfois jusqu'à 32767, si vous disposez d'un RAM de 16K). Tout comme avec le ROM, vous pouvez trouver les valeurs de ces bytes en enfonçant **PEEK**, mais la différence avec ROM est que vous pouvez également les changer. (Le ROM, en effet, est fixe et ne peut pas être changé).

Ecrivez:

POKE 17300,57

Le byte sur l'adresse 17300 reçoit ainsi la valeur 57. Si vous écrivez maintenant:

PRINT PEEK 17300

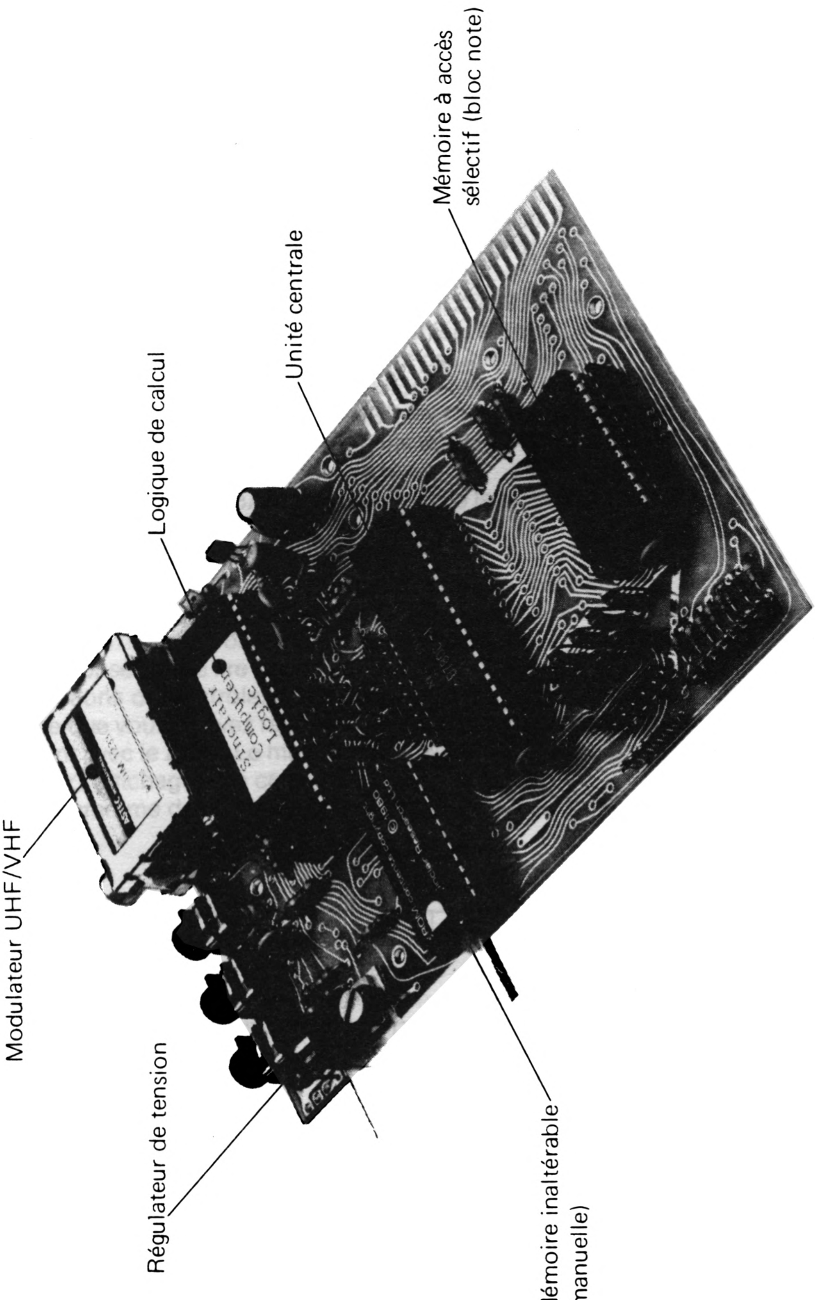
vous verrez apparaître le chiffre 57.

Remarquez que l'adresse doit se situer entre 0 et 65535, et la plupart de celles-ci se réfèrent à des bytes. La valeur doit se situer entre -255 et +255, et quand elle est négative, 256 est ajouté. La capacité **POKE**, si vous savez comment l'utiliser, vous donne une énorme puissance sur l'ordinateur. Néanmoins, la connaissance nécessaire n'est pas à apprendre dans un manuel d'introduction comme celui-ci.

Le dernier chip important est le chip logique ou le SCL (Sinclair Computer Logic) chip. Ce chip est également conçu spécialement pour le ZX81 et il relie tous les autres chips d'une façon extraordinaire. Ainsi, ils peuvent réaliser beaucoup plus que pris à part.

Le modulateur convertit le **OUTPUT** de l'écran en une forme qui convient pour un téléviseur et le régulateur convertit les 9 volts de la source d'énergie en 5 volts.

Sommaire:
Chips.
Déclaration: **POKE**.
Fonction: **PEEK**.


Modulateur UHF/VHF
Régulateur de tension
Logique de calcul
Unité centrale
Mémoire à accès sélectif (bloc note)
Mémoire inaltérable (manuelle)

L'EMPLOI DU CODE MACHINE

Ce chapitre est écrit pour ceux qui comprennent le code machine Z8Ø, c.-à-d. l'ensemble des instructions employées par le microprocesseur de commande de l'unité centrale Z8Ø. Si vous ne le comprenez pas, mais que vous aimeriez en savoir plus, consultez des livres qui traitent de ce sujet. Les deux livres d'introduction que nous vous recommandons sont: "Programming the ZX81" de Rodney Zaks, publié par Sybex au prix de 10 livres environ, et "Z8Ø et 8Ø8Ø Assembly Language Programming" de Rathe Sprancklen, publié par Hayden au prix de 5,25 livres.

Les publications faisant autorité à ce sujet sont: "Z8Ø Asembly Language Programming Manual" et "Z8Ø - CPU, Z8Ø A-CPU Technical Manual", édité par Zilog, à 6 livres environ pour les deux manuels. Mais ils ne sont pas à recommander aux débutants. Les routines de code machine peuvent être exécutées à partir d'un programme BASIC, utilisant la fonction **USR**. L'argument de **USR** est l'adresse de départ de la routine et le résultat est un chiffre entier de deux bytes, le contenu de la paire de registres bc au retour.

L'adresse de retour vers le BASIC est stockée de la façon habituelle, le retour se fait donc par une instruction RET du Z8Ø.

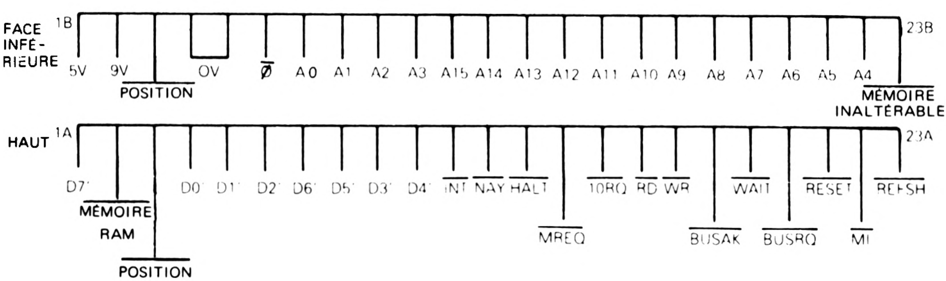
Il y a quelques restrictions aux routines **USR**:

(i) au retour, les registres iy et i doivent avoir les valeurs 4ØØØ et 1Eh.

(ii) la routine d'affichage utilise les registres a', f', iy, i et r. Donc la routine **USR** ne peut pas les employer au moment où les opérations de calcul et d'affichage sont en cours. (Il est même à déconseiller de lire la paire af').

Les connections de contrôle, de données et d'adresses sont affichées au dos du ZX81. Tout ce que vous pouvez faire avec le ZX8Ø, vous pouvez donc pratiquement le faire également avec le ZX81. Le matériel (hardware) ZX81 peut parfois vous gêner, en particulier pour l'affichage et le calcul.

Voici un diagramme des connections qui figurent sur le dos de l'ordinateur



Une partie de ce code machine, se trouvant au milieu de la mémoire, court le risque d'être effacé par le système BASIC.

Mais, il y a des endroits plus sûrs:
(i) Dans une déclaration **REM**. Ecrivez une déclaration **REM** qui possède assez de caractères pour contenir votre code machine, que vous introduisez. Faites-en la première ligne du programme, sinon vous risquez de la voir déménager ensuite. Evitez les instructions "halte" qui sont perçues comme la fin de la déclaration **REM**.
(ii) Dans une chaîne. Vous créez une chaîne suffisamment longue, et vous affectez un byte de code machine à chaque caractère. Les chaînes se présentent toujours à un déplacement dans la mémoire.

Dans l'annexe A, l'ensemble des caractères, vous trouverez les caractères et les instructions Z8Ø rangées côte à côte et en ordre. L'utilité vous en paraîtra lorsque vous introduisez le code.
(iii) Au sommet de la mémoire, quand le ZX81 est branché, il contrôle de combien de mémoire vous disposez et place le 'tas' de commande au sommet de sorte qu'il n'y ait là plus de place pour des routines **USR**. Il emmagasine l'adresse du premier byte non-existant (par ex. 17K ou 174Ø8, si vous avez une mémoire de 1K) dans une variable de système dénommé RAMPTOP, dans les deux bytes dont les adresses sont 16388 et 16389. **NEW** d'autre part ne fait pas de contrôle total de la mémoire, mais un contrôle limité s'arrêtant juste avant l'adresse dans RAMTOP.

Donc, si vous introduisez l'adresse d'un byte existant dans RAMTOP, une déclaration **NEW** aura comme effet que toute la mémoire, à partir de ce byte, sera en dehors du système BASIC et donc isolée. Si, par ex., vous avez une mémoire de 1K et que vous venez d'allumer l'ordinateur

PRINT PEEK 16388 + 256 **PEEK** 16389

vous indique l'adresse (174Ø8) du premier byte non-existant.

Supposons maintenant que vous avez une routine **USR** de 2Ø bytes. Vous désirez changer RAMTOP en 17388 = 236 + 256 67 (comment procéderez-vous avec l'ordinateur?) Ecrivez:

POKE 16388,236
POKE 16389,67

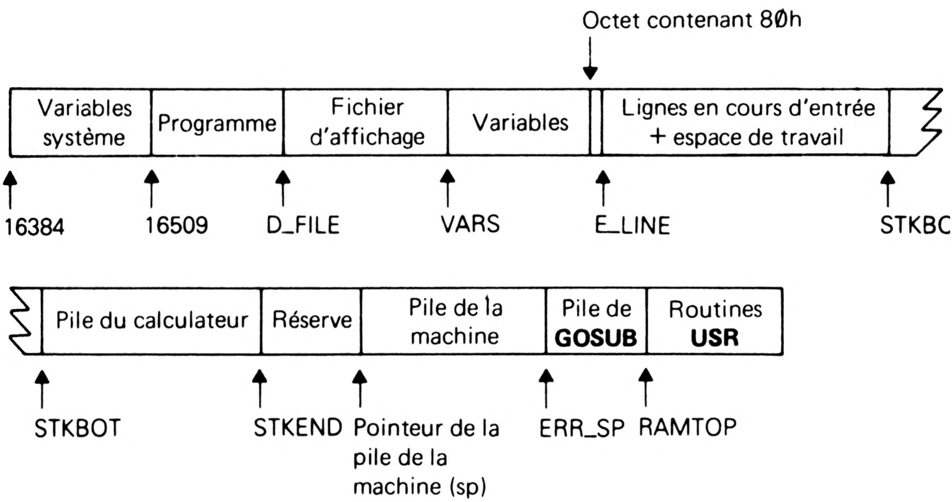
et ensuite **NEW**. Vous pouvez maintenant faire ce que vous voulez des 2Ø bytes de mémoire à partir de l'adresse 17388 jusqu'à l'adresse 174Ø7.
Si vous récrivez **NEW**, cela n'effacera pas ces 2Ø bytes. Le sommet de la mémoire est une bonne place pour les routines **USR**, sûre et immobile.
Son inconvénient majeur est qu'il n'est pas protégé par **SAVE**.

Sommaire:
Fonction: **USR**.
Déclaration: **NEW**.

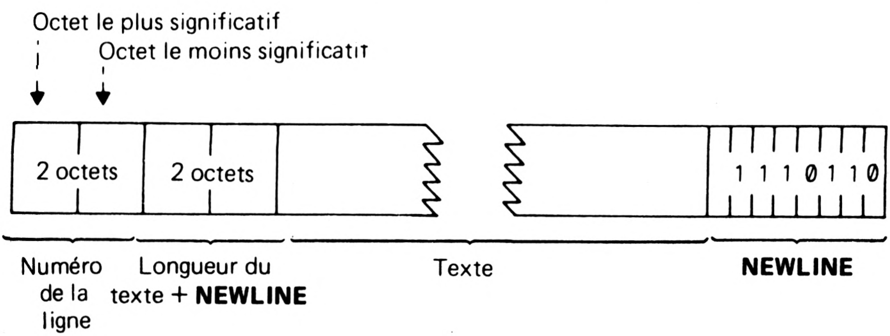
Exercice:
1. Faites un RAMTOP égal à 167ØØ et exécutez **NEW**. Vous vous ferez une idée de ce qui se passe quand la mémoire est remplie.

L'ORGANISATION DE LA MEMOIRE

La mémoire est divisée en différentes zones pour emmagasiner les différentes sortes d'informations. Ces zones sont juste assez grandes pour l'information qu'elles contiennent, et si vous insérez de l'information supplémentaire, à un certain moment, (par ex. en ajoutant une ligne de programme ou une variable) vous faites de la place en décalant le tout au-dessus de ce point. Inversément, si vous éliminez de l'information, tout ce qui se trouve au-dessus, est déplacé vers le bas.



Les variables de système contiennent l'information qui indique à l'ordinateur en quel état il se trouve. Elles sont toutes reprises dans le prochain chapitre, mais pour l'instant sachez qu'il y en a (les fichiers D, VARS, ligne—E, etc.) qui contiennent les adresses et les bornes entre les différentes zones de la mémoire. Ce ne sont pas des variables du BASIC, et leurs noms ne seront pas reconnus par l'ordinateur.
Chaque ligne de programme est emmagasinée comme décrit ci-dessous:



Remarquez que, en contradiction avec les autres cas de numéros de deux bytes, le numéro de ligne (comme pour la variable de contrôle **FOR-NEXT**) est emmagasiné, avec son byte le plus significatif en première place, c.-à-d. dans l'ordre que vous l'introduisez.

Une constante numérique du programme est suivie par sa forme binaire, employant le caractère **CHR$** 126 suivi par 5 autres bytes pour le numéro même.
Le fichier d'affichage est la copie mémoire de l'image télévisée. Il commence par un caractère **NEWLINE** et ensuite il y a 24 lignes de texte qui se terminent toutes par **NEWLINE**.

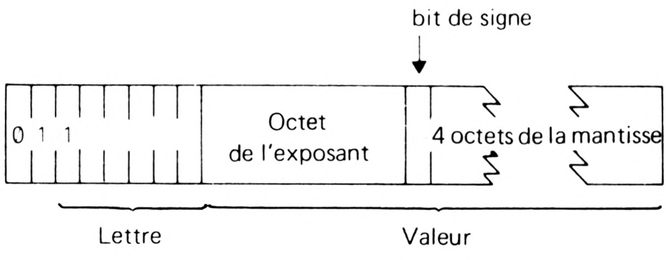
Le système est conçu de telle façon qu'une ligne de texte ne doit pas nécessairement remplir l'espace de 32 caractères. Les espaces finaux peuvent être omis. Cela sert à économiser de l'espace quand la mémoire est petite.

Quand toute la mémoire est plus petite que 3 1/4 K (ce qui dépend de la variable de système RAMTOP), l'écran vierge – comme il est donné au début – contient exactement 25 **NEWLINE**s.

Quand la mémoire est plus grande, l'écran vierge est agrandi à 24*32 espaces et en pratique, il conserve entièrement son format.

Néanmoins, **SCROLL**, et certaines conditions où les lignes de fond s'étendent à plus de deux lignes, peut bouleverser toute la situation en introduisant quelques lignes courtes en bas de l'écran. Les variables ont des formats différents, selon leurs natures différentes.

Nombre dont le nom est composé d'une seule lettre:



Nombre dont le nom comporte plus d'une lettre:

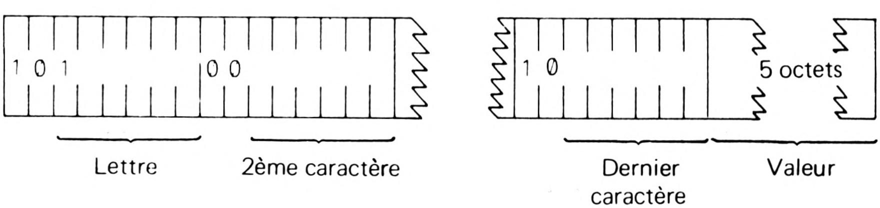
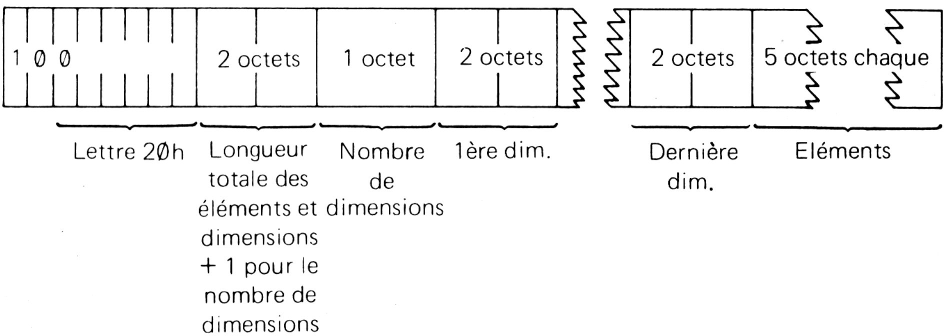


Tableau des nombres

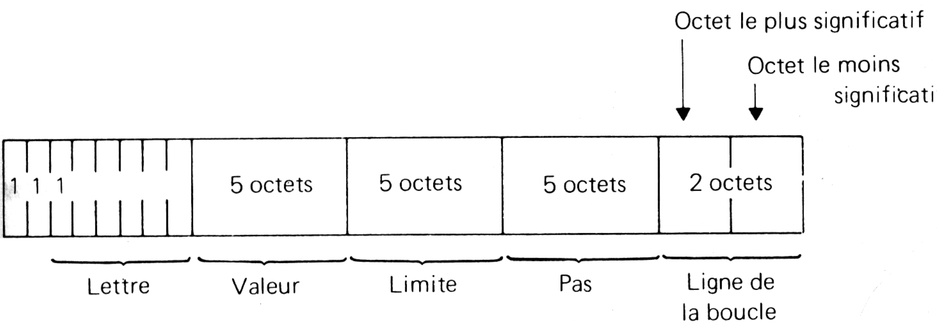


Voici l'ordre des éléments:
d'abord les éléments dont le premier indice est 1;
ensuite les éléments dont le premier indice est 2;
et alors les éléments dont le premier indice est 3
etc. pour toutes les valeurs possibles des premiers indices.

Les éléments dont les premiers indices sont identiques, sont classés - suivant le même système - selon le second indice, etc. jusqu'au dernier indice.

Exemple: Les éléments du tableau B 3 x 6 du chapitre 22 sont emmagasinés selon l'ordre suivant: B(1,1), B(1,2), B(1,3), B(1,4), B(1,5), B(1,6), B(2,1), B(2,2),...B(2,6), B(3,1),...B(3,6).

Variable de contrôle d'une boucle **FOR-NEXT** :



Chaîne :

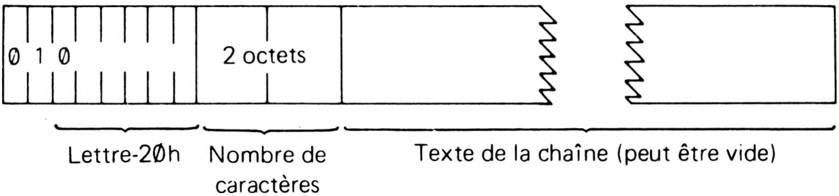
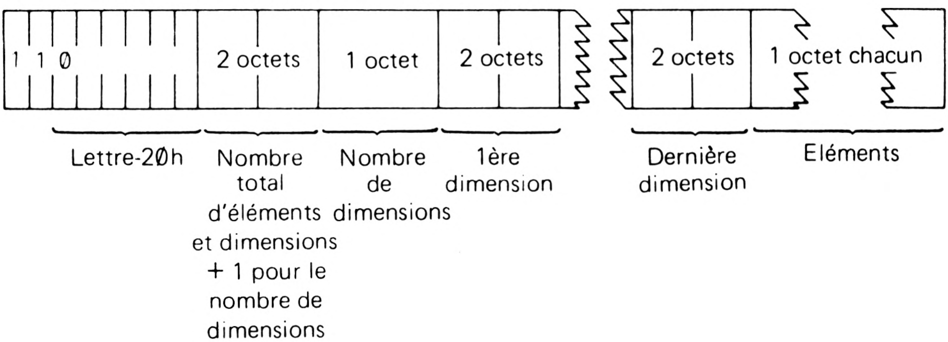


Tableau de caractères :



La partie commençant par E—LINE contient la ligne écrite à ce moment précis, (comme une instruction, une ligne de programme, ou une donnée **INPUT**) et, en plus un peu d'espace de travail.

Le calculateur est la partie du système BASIC qui s'occupe de l'arithmétique et la plupart des chiffres sur lesquels il opère sont emmagasinés dans le 'tas' du calculateur.

La réserve contient l'espace qui n'est pas encore employé.

Le 'tas'-machine est le tas qui est employé par le chip Z80 pour contenir les adresses de retour etc.

Le 'tas' **GOSUB** a été décrit au chapitre 14.

La zone pour les routines **USR** doit être prévue par vous-même, en employant **NEW** (Cfr. le dernier chapitre).

LES VARIABLES DU SYSTEME

Les bytes en mémoire de 16384 à 165Ø8 sont mis à part par le système pour des fonctions spécifiques. Enfoncez **PEEK** afin de recevoir plus d'explications sur ce système et n'oubliez pas **POKE**. En faisant cela, ils seront affichés dans une liste avec tous leurs usages.

C'est ce qu'on appelle les variables de système, et elles ont des noms. Ne les confondez pas avec les variables du **BASIC**. L'ordinateur ne les comprendra pas comme étant des variables de système, et ils sont seulement donnés en tant que mnémonique pour les hommes.

Les abréviations de la colonne 1 ont les spécifications suivantes:

X: n'employez pas **POKE**, car le système pourrait se bloquer;
N: **POKE** n'aura pas un effet satisfaisant sur les variables;
S: la variable est conservée par **SAVE**.

Le chiffre dans la première colonne indique le nombre de bytes que contient la variable. Quand il y a deux bytes, le premier est le moins sigificatif, contrairement à ce que vous pourriez croire. Si vous voulez appliquer **POKE**, pour une valeur v, sur une variable qui consiste de deux bytes, à l'adresse n, faites comme montré ci-dessous:

POKE n,v - 256***INT** (v/256)
POKE n + 1, **INT** (v/256)

Si vous voulez y appliquer une fonction **PEEK1:**

PEEK n + 256***PEEK** (n + 1)

Note	*Adresse*	*Nom*	*Contenu*
1	16384	ERR-NR	1 de moins que le code de rapport. Commencez à 255 (pour-1) donc **PEEK** 16384, si cela marche, ça donne 255. **POKE** 16384,n peut être utilisé pour forcer une halte d'erreur Ø n 14 donne un rapport usuel, 15 n 34 ou 99 n 127 donne un rapport spécial. 35 n 98 causera confusion dans la liste d'affichage.
X1	16385	FLAGS	Différents indicateurs pour contrôler le BASIC.
X2	16386	ERR-SP	Adresse du 1° élément du 'tas'-machine (après retour **GOSUB**).

Note	*Adresse*	*Nom*	*Contenu*
2	16388	RAMTOP	Adresse du 1° byte au-dessus de la zone du système BASIC. Enfoncez **POKE** pour faire une nouvelle zone de réserve (**NEW**) au-dessus de cette zone (Cfr. chap. 16), ou pour duper **CLS** en créant une liste minimale d'affichage... **POKE** n'a aucun effet sur RAMTOP aussi longtemps qu'un de ces deux ne soit pas exécuté.
N1	16390	MODE	Spécifie les curseurs K, L, F ou G.
N2	16391	PPC	Numéro de ligne de la déclaration exécutée à un certain moment. **POKE** n'a aucun effet durable, sauf à la dernière ligne du programme.
S1	16393	VERSN	Ø identifie le BASIC du ZX81 dans les programmes mémorisés.
S2	16394	E-PPC	Numéro de ligne courante (avec le curseur du programme).
SX2	16396	D-FILE	Cfr. chapitre 27.
S2	16398	DF-CC	Adresse de la position **PRINT** dans la liste d'affichage. **POKE** envoie le **OUTPUT PRINT** autre part.
SX2	164ØØ	VARS	Cfr. chapitre 27.
SN2	164Ø2	DEST	Adresse de la variable dans une déclaration.
SX2	164Ø4	E-LINE	Cfr. chapitre 27.
SX2	164Ø6	CH-ADD	Adresse du prochain caractère à imprimer. Le caractère après l'argument **PEEK**, ou **NEWLINE** après une déclaration **POKE**.
S2	164Ø8	X-PTR	Adresse du caractère suivi par le curseur
SX2	1641Ø	STKBOT	Cfr. chapitre 27.
SX2	16412	STKEND	
SN1	16414	BERG	Registre b du calculateur.
SN2	16415	MEM	Adresse pour la zone du mémoire du calculateur (en principe **MEMBOT**, mais pas toujours).
S1	16417		PAS EMPLOYE
SX1	16418	DF-SZ	Nombre de lignes (une ligne vierge incluse) du fond de l'écran.
S2	16419	S-TOP	Numéro de la 1° ligne d'un programme dans les listages automatiques.
SN2	15421	LAST-K	Indique la touche enfoncée.
SN1	16423		

Note	*Adresse*	*Nom*	*Contenu*
SN1	16424	MARGIN	Nombre de lignes vierges en-dessous et au-dessus de l'image (55 en B.B et 31 aux E.U.)
SX2	16425	NXTLIN	Adresse de la prochaine ligne du programme à exécuter.
S2	16427	OLDPPC	Numéro de ligne de laquelle **CONT** saute.
SN1	16429	FLAGX	De nombreux indicateurs.
SN2	16430	STRLEN	Longueur d'une chaîne affectée.
SN2	16432	T-ADDR	Adresse du prochain élément de la table de syntaxe (pas utile).
S2	16434	SEED	Seed pour **RND**. C'est la variable créée par **RAND**.
S2	16436	FRAMES	Compte les images télévisées. Bit 15 est 1. Bits entre 0 et 14 sont diminués pour chaque image télévisée. Ceci peut servir de synchronisation, mais **PAUSE** a le même effet. **PAUSE** replace bit 15 à 0, et insère les bits de 0 à 14 dans l'espace de la **PAUSE**. Quand ceux-ci sont décomptés jusqu'à 0, la pause s'arrête. Si, en enfonçant une touche, la pause s'arrête, le bit 15 est remis à 1.
S1	15438	COORDS	Coordonnée X du dernier point **PLOT**.
S1	16439		Coordonnée Y du dernier point **PLOT**.
S1		PR-CC	Byte moins significatif de l'adresse de la prochaine position de **LPRINT** à imprimer (en (**PRBUFF**).
SX1	16441	S-POSN	Numéro de colonne de la position **PRINT**.
S1	16442		Numéro de ligne pour la position **PRINT**.
S1	16443	CDFLAGS	De nombreux indicateurs. Bit 7 est à (1) durant le mode de calcul et d'affichage.
S33	16444	PRBUFF	Zone tampon de l'imprimante (le 33ième caractère est **NEWLINE**).
SN30	16477	MEMBOT	Zone de la mémoire du calculateur. Sert à emmagasiner les chiffres qui ne peuvent pas être entassés sur le tas (stack) du calculateur.
S2	16507		PAS EMPLOYE

Exercices:

1. Essayez ce programme-ci:

```
10 FOR N = 0 TO 21
20 PRINT PEEK (PEEK 16400 + 256*PEEK 16401 + N)
30 NEXT N
```

Il vous donne les 22 premiers bytes de la zone des variables. Essayez de mettre en concordance la variable de contrôle N avec la description du chapitre 27.

2. Changez la ligne 20 du programme ci-dessus en:

```
20 PRINT PEEK (16509 + N)
```

Ceci vous donne les 22 premiers bytes de la zone de programme. Concordez-le avec le programme même.

L'ENSEMBLE DES CARACTERES

Voici tout l'ensemble de caractères du ZX81, avec leur code en système décimal et en système HEX. Si nous imaginons les codes comme étant des codes d'instruction de machine Z8Ø, la colonne de droite vous donne le langage d'assemblage mnémonique correspondant. Si vous comprenez ces choses-là, vous savez que certaines instructions Z8Ø sont des composantes commençant par CBh ou EDh. Les deux colonnes de droite vous les donnent.

Code	*Caractère*	*Hex*	*Assembleur Z80*	*– après CBh*
0	espace	00	nop	rlc b
1	▘	01	ld bc,NN	rlc c
2	▝	02	ld (bc),a	rlc d
3	▀	03	inc bc	rlc e
4	▖	04	inc b	rlc h
5	▌	05	dec b	rlc l
6	▞	06	ld b,N	rlc (hl)
7	▛	07	rlca	rlc a
8	▒	08	ex af,af'	rrc b
9	🮏	09	add hl,bc	rrc c
10	🮎	0A	ld a,(bc)	rrc d
11	"	0B	dec bc	rrc e
12	£	0C	inc c	rrc h
13	$	0D	dec c	rrc l
14	:	0E	ld, c,N	rrc (hl)
15	?	0F	rrca	rrc a
16	(	10	djnz DIS	rl b
17	)	11	ld de,NN	rl c
18	>	12	ld (de),a	rl d
19	<	13	inc de	rl e
20	=	14	inc d	rl h
21	+	15	dec d	rl l
22	–	16	ld d,N	rl (hl)
23	*	17	rla	rl a
24	/	18	jr DIS	rr b
25	;	19	add hl,de	rr c
26	,	1A	ld a,(de)	rr d
27	.	1B	dec de	rr e
28	0	1C	inc e	rr h
29	1	1D	dec e	rr l
30	2	1E	ld e,N	rr (hl)
31	3	1F	rra	rr a
32	4	20	jr nz,DIS	sla b
33	5	21	ld hl,NN	sla c
34	6	22	ld (NN),hl	sla d

Code	*Caractère*	*Hex*	*Assembleur Z80*	*– après CBh*	*– après EDh*
35	7	23	inc hl	sla e	
36	8	24	inc h	sla h	
37	9	25	dec h	sla l	
38	A	26	ld, h,N	sla (hl)	
39	B	27	daa	sla a	
40	C	28	jr z,DIS	sra b	
41	D	29	add hl,hl	sra c	
42	E	2A	ld hl,(NN)	sra d	
43	F	2B	dec hl	sra e	
44	G	2C	inc l	sra h	
45	H	2D	dec l	sra l	
46	I	2E	ld,l,N	sra (hl)	
47	J	2F	cpl	sra a	
48	K	30	jr nc,DIS		
49	L	31	ld sp,NN		
50	M	32	ld (NN),a		
51	N	33	inc sp		
52	O	34	inc (hl)		
53	P	35	dec (hl)		
54	Q	36	ld (hl),N		
55	R	37	scf		
56	S	38	jr c,DIS	srl b	
57	T	39	add hl,sp	srl c	
58	U	3A	ld a,(NN)	srl d	
59	V	3B	dec sp	srl e	
60	W	3C	inc a	srl h	
61	X	3D	dec a	srl l	
62	Y	3E	ld a,N	srl (hl)	
63	Z	3F	ccf	srl a	
64	**RND**	40	ld b,b	bit 0,b	in b,(c)
65	**INKEY$**	41	ld b,c	bit o,c	out (c),b
66	**PI**	42	ld b,d	bit 0,d	sbc hl,bc
67		43	ld b,e	bit 0,e	ld (NN),bc
68		44	ld b,h	bit 0,h	neg
69		45	ld b,l	bit 0,l	retn
70		46	ld b,(hl)	bit 0,(hl)	im 0
71		47	ld b,a	bit 0,a	ld i,a
72	} inutilisés	48	ld c,b	bit 1,b	in c,(c)
73		49	ld c,c	bit 1,c	out (c),c
74		4A	ld c,d	bit 1,d	adc hl,bc
75		4B	ld c,e	bit 1,e	ld bc,(NN)
76		4C	ld c,h	bit 1,h	

Code	*Caractère*	*Hex*	*Assembleur Z80*	*– après CBh*	*– après EDh*
77		4D	ld c,l	bit 1,l	reti
78		4E	ld c,(hl)	bit 1,(hl)	
79		4F	ld c,a	bit 1,a	ld r, a
80		50	ld d,b	bit 2,b	in d,(c)
81		51	ld d,c	bit 2,c	out (c),d
82		52	ld d,d	bit 2,d	sbc hl,de
83		53	ld d,e	bit 2,e	ld (NN),de
84		54	ld d,h	bit 2,h	
85		55	ld d,l	bit 2,l	
86		56	ld d,(hl)	bit 2,(hl)	im 1
87		57	ld d,a	bit 2,a	ld a,i
88		58	ld e,b	bit 3,b	in e,(c)
89		59	ld e,c	bit 3,c	out (c),e
90		5A	ld e,d	bit 3,d	adc hl,de
91		5B	ld e,e	bit 3,e	ld de,(NN)
92		5C	ld e,h	bit 3,h	
93		5D	ld e,l	bit 3,l	
94	inutilisés	5E	ld e,(hl)	bit 3,(hl)	im 2
95		5F	ld e,a	bit 3,a	ld a, r
96		60	ld h,b	bit 4,b	in h,(c)
97		61	ld h,c	bit 4,c	out (c),h
98		62	ld h,d	bit 4,d	sbc hl,hl
99		63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103		67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld l,a	bit 5,a	rld
112	montée curseur ⇧	70	ld (hl),b	bit 6,b	
113	descente curseur ⇩	71	ld (hl),c	bit 6,c	
114	curs. vers la gauche ⇦	72	ld (hl),d	bit 6,d	sbc hl,sp
115	curs. vers la droite ⇨	73	ld (hl),e	bit 6,e	ld (NN),sp
116	**GRAPHICS**	74	ld (hl),h	bit 6,h	
117	**EDIT**	75	ld (hl),l	bit 6,l	
118	**NEWLINE**	76	halt	bit 6,(hl)	

Code	*Caractère*	*Hex*	*Assembleur Z80*	*– après CBh*	*– après EDh*
119	**RUBOUT**	77	ld (hl),a	bit 6,a	
120	mode 🄺 \ 🄻	78	ld a,b	bit 7,b	in a,(c)
121	**FUNCTION**	79	ld a,c	bit 7,c	out (c),a
122	inutilisé	7A	ld a,d	bit 7,d	adc hl,sp
123	inutilisé	7B	id a,e	bit 7,e	ld sp,(NN)
124	inutilisé	7C	ld a,h	bit 7,h	
125	inutilisé	7D	ld a,l	bit 7,l	
126	nombre	7E	ld a,(hl)	bit 7,(hl)	
127	curseur	7F	ld a,a	bit 7,a	
128	█	80	add a,b	res 0,b	
129	▟	81	add a,c	res 0,c	
130	▙	82	add a,d	res 0,d	
131	▄	83	add a,e	res 0,e	
132	▜	84	add a,h	res 0,h	
133	▐	85	add a,l	res 0,l	
134	▚	86	add a,(hl)	res 0,(hl)	
135	▛	87	add a,a	res 0,a	
136	▒	88	adc a,b	res 1,b	
137	▀▒	89	adc a,c	res 1,c	
138	▒▄	8A	adc a,d	res 1,d	
139	" inversé	8B	adc a,e	res 1,e	
140	£ inversé	8C	adc a,h	res 1,h	
141	$ inversé	8D	adc a,l	res 1,l	
142	: inversé	8E	adc a,(hl)	res 1,(hl)	
143	? inversé	8F	adc a,a	res 1,a	
144	(inversé	90	sub b	res 2,b	
145	) inversé	91	sub c	res 2,c	
146	> inversé	92	sub d	res 2,d	
147	< inversé	93	sub e	res 2,e	
148	= inversé	94	sub h	res 2,h	
149	+ inversé	95	sub l	res 2,1	
150	– inversé	96	sub (hl)	res 2,(hl)	
151	* inversé	97	sub a	res 2,a	
152	/ inversé	98	sbc a,b	res 3,b	
153	; inversé	99	sbc a,c	res 3,c	
154	, inversé	9A	sbc a,d	res 3,d	
155	. inversé	9B	sbc a,e	res 3,e	
156	0 inversé	9C	sbc a,h	res 3,h	
157	1 inversé	9D	sbc a,l	res 3,l	
158	2 inversé	9E	sbc a,(hl)	res 3,(hl)	
159	3 inversé	9F	sbc a,a	res 3,a	
160	4 inversé	A0	and b	res 4,b	ldi

Code	*Caractère*	*Hex*	*Assembleur Z80*	*– après CBh*	*– après EDh*
161	5 inversé	A1	and c	res 4,c	cpi
162	6 inversé	A2	and d	res 4,d	ini
163	7 inversé	A3	and e	res 4,e	outi
164	8 inversé	A4	and h	res 4,h	
165	9 inversé	A5	and !	res 4,1	
166	A inversé	A6	and (hl)	res 4,(hl)	
167	B inversé	A7	and a	res 4,a	
168	C inversé	A8	xor b	res 5,b	ldd
169	D inversé	A9	xor c	res 5,c	cpd
170	E inversé	AA	xor d	res 5,d	ind
171	F inversé	AB	xor e	res 5,e	outd
172	G inversé	AC	xor h	res 5,h	
173	H inversé	AD	xor l	res 5,l	
174	I inversé	AE	xor (hl)	res 5,(hl)	
175	J inversé	AF	xor a	res 5,a	
176	K inversé	B0	or b	res 6,b	ldir
177	L inversé	B1	or c	res 6,c	cpir
178	M inversé	B2	or d	res 6,d	inir
179	N inversé	B3	or e	res 6,e	otir
180	O inversé	B4	or h	res 6,h	
181	P inversé	B5	or l	res 6,l	
182	Q inversé	B6	or (hl)	res 6,(hl)	
183	R inversé	B7	or a	res 6,a	
184	S inversé	B8	cp b	res 7,b	lddr
185	T inversé	B9	cp c	res 7,c	cpdr
186	U inversé	BA	cp d	res 7,d	indr
187	V inversé	BB	cp e	res 7,e	otdr
188	W inversé	BC	cp h	res 7,h	
189	X inversé	BD	cp l	res 7,l	
190	Y inversé	BE	cp (hl)	res 7,(hl)	
191	Z inversé	BF	cp a	res 7,a	
192	" "	C0	ret nz	set 0,b	
193	**AT**	C1	pop bc	set 0,c	
194	**TAB**	C2	jp nz,NN	set 0,d	
195	inutilisé	C3	jp NN	set 0,e	
196	**CODE**	C4	call nz,NN	set 0,h	
197	**VAL**	C5	push bc	set 0,l	
198	**LEN**	C6	add a,N	set 0,(hl)	
199	**SIN**	C7	rst 0	set 0,a	
200	**COS**	C8	ret z	set 1,b	
201	**TAN**	C9	ret	set 1,c	
202	**ASN**	CA	jp z,NN	set l,d	

Code	*Caractère*	*Hex*	*Assembleur Z80*	*— après CBh*
203	**ACS**	CB		set l,e
204	**ATN**	CC	call z,NN	set 1,h
205	**LN**	CD	call NN	set 1,l
206	**EXP**	CE	adc a,N	set 1,(hl)
207	**INT**	CF	rst 8	set 1,a
208	**SQR**	D0	ret nc	set 2,b
209	**SGN**	D1	pop de	set 2,c
210	**ABS**	D2	jp nc,NN	set 2,d
211	**PEEK**	D3	out N,a	set 2,e
212	**USR**	D4	call nc,NN	set 2,h
213	**STR$**	D5	push de	set 2,l
214	**CHR$**	D6	sub N	set 2,(hl)
215	**NOT**	D7	rst 16	set 2,a
216	**	D8	ret c	set 3,b
217	**OR**	D9	exx	set 3,c
218	**AND**	DA	jp c,NN	set 3,d
219	<=	DB	in a,N	set 3,e
220	>=	DC	call c,NN	set 3,h
221	<>	DD	préfixe les instructions avec ix	set 3,1
222	**THEN**	DE	sbc a,N	set 3,(hl)
223	**TO**	DF	rst 24	set 3,a
224	**STEP**	E0	ret po	set 4,b
225	**LPRINT**	E1	pop hl	set 4,c
226	**LLIST**	E2	jp po,NN	set 4,d
227	**STOP**	E3	ex (sp),hl	set 4,e
228	**SLOW**	E4	call po,NN	set 4,h
229	**FAST**	E5	push hl	set 4,l
230	**NEW**	E6	and N	set 4,(hl)
231	**SCROLL**	E7	rst 32	set 4,a
232	**CONT**	E8	ret pe	set 5,b
233	**DIM**	E9	jp (hl)	set 5,c
234	**REM**	EA	jp pe,NN	set 5,d
235	**FOR**	EB	ex de,hl	set 5,e
236	**GOTO**	EC	call pe,NN	set 5,h
237	**GOSUB**	ED		set 5,l
238	**INPUT**	EE	xor N	set 5,(hl)
239	**LOAD**	EF	rst 40	set 5,a
240	**LIST**	F0	ret p	set 6,b
241	**LET**	F1	pop af	set 6,c
242	**PAUSE**	F2	jp p,NN	set 6,d
243	**NEXT**	F3	di	set 6,e

Code	*Caractère*	*Hex*	*Assembleur Z80*	*— après CBh*
244	**POKE**	F4	call p,NN	set 6,h
245	**PRINT**	F5	push af	set 6,l
246	**PLOT**	F6	or N	set 6,(hl)
247	**RUN**	F7	rst 48	set 6,a
248	**SAVE**	F8	ret m	set 7,b
249	**RAND**	F9	ld sp,hl	set 7,c
250	**IF**	FA	jp m,NN	set 7,d
251	**CLS**	FB	ei	set 7,e
252	**UNPLOT**	FC	call m,NN	set 7,h
253	**CLEAR**	FD	préfixe les ins-turctions avec iy	set 7,l
254	**RETURN**	FE	cp N	set 7,(hl)
255	**COPY**	FF	rst 56	set 7,a

LES CODES DE RAPPORT

Cette table vous donne tous les rapports avec une description générale et une liste de situations dans lesquelles ils peuvent apparaître. Dans l'annexe C, vous trouverez une description plus détaillée de la signification des rapports d'erreurs sous chaque déclaration ou fonction.

Code	*Signification*	*Situation*
Ø	Accomplissement réussi ou saut vers un numéro de ligne plus haut qu'il n'existe. Un code avec rapport Ø ne change pas le numéro de ligne employé par **CONT**.	Peu importe.
1	La variable de contrôle n'existe pas (n'a pas été introduite par une déclaration **SET**), mais il y a une variable ordinaire du même nom.	**NEXT**
2	Une variable non-identifiée a été employée. Cela se passe avec de simples variables si la variable est employée avant d'être affectée dans une déclaration **LET**. Cela se passe avec des variables indicées si la variable est employée avant d'avoir reçu une dimension dans une déclaration **DIM**, s'il n'y a pas de variables ordinaires du même nom.	Peu importe.
3	Indice hors de portée. S'il est désespérément hors de portée (neg. ou plus grand que 65535), il y aura l'erreur B.	Variables indicées.
4	Pas assez de place dans la mémoire. Remarquez que le numéro de ligne dans le rapport (après /) peut apparaître de façon incomplète à l'écran à cause d'un manque de mémoire. Par ex. 4/2Ø peut apparaître comme 4/2 (Cfr. chap. 23). Pour **GOSUB**: Cfr. ex. 6 du chap. 14.	**LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB.** Quelquefois pendant l'évaluation des fonctions.
5	Plus de place à l'écran, **CONT** fera de la place en nettoyant l'écran.	**RINT** et **LIST**.
6	Un trop-plein arithmétique. Les calculs résultent en un chiffre supérieur à $1Ø^{38}$.	Peu importe.
7	Pas de **GOSUB** correspondant à une déclaration **RETURN**.	**RETURN**.
8	Vous avez employé **INPUT** comme une instruction (c'est interdit).	**INPUT**.
9	Déclaration **STOP** exécutée. La déclaration **CONT** n'essaiera pas de ré-exécuter la déclaration **STOP**.	**STOP**.

Code	*Signification*	*Situation*
A	Argument qui n'est pas valable pour certaines fonctions.	**SQR, LN, ASN, ACS**.
B	Chiffre entier hors de portée. Quand un chiffre entier est requis, l'argument de la virgule flottante est arrondi vers le chiffre entier le plus proche. Si c'est en-dehors d'une portée admissible, vous recevez l'erreur B. Pour l'accès aux tableaux, cfr. rapport 3.	**RUN, RAND, POKE, DIM, GOTO, GOSUB, LIST, LLIST, POKE, PLOT, UNPLOT, CHR$, PEEK, USR.**
C	Le texte d'un argument (de chaîne) de **VAL**, ne forme pas une expression valable.	**VAL**.
D	(i) programme interrompu par **BREAK**.	A la fin de n'importe quelle déclaration, ou avec **LOAD, SAVE, LPRINT, LLIST,** ou **COPY**.
	(ii) la ligne input commence par **STOP**	**INPUT**.
E	Pas employé.	
F	Le nom du programme fourni est la chaîne vide.	**SAVE**.

LE ZX81 POUR CEUX QUI CONNAISSENT LE BASIC

En général:
Si vous connaissez le BASIC, vous ne devriez pas avoir de problèmes avec le ZX81. Mais, il y a tout de même une ou deux idiosyncrasies.

(i) Les mots ne sont pas entièrement épelés, mais ils disposent de leurs propres mots-clé – ce qui est expliqué au chapitre 2 (pour les mots-clés et les touches qu'on doit employer simultanément avec **SHIFT**) et le chapitre 5 (pour les noms de fonctions). Ces mots seront imprimés en gras dans le texte qui suit.

(ii) Le BASIC du ZX81 manque **READ, DATA** et **RESTORE** (Cfr. chap. 22, ex. 3), les fonctions définies par l'utilisateur même (**FN** et **DEF**, mais **VAL** peut parfois être employé) et des lignes avec des multiples déclarations.

(iii) Les facilités d'emploi de chaînes sont très étendues mais non-standardisées. (Cfr. chap. 21 et 22 pour les tableaux de chaînes.).

(iv) L'ensemble de caractères est propre au ZX81.

(v) L'affichage télévisé n'est généralement pas mémorisé.

(vi) Si vous êtes accoutumés à **PEEK** et **POKE** sur d'autres machines, sachez que toutes les adresses seront différentes sur le ZX81.

La vitesse:
L'ordinateur a deux vitesses: le mode d'affichage et calcul,
le mode rapide.

Dans le mode d'affichage et calcul vous recevez continuellement une image et le calcul se fait pendant les périodes où l'écran est blanc.

Dans le mode rapide, vous ne recevez pas d'image télévisée pendant le calcul.

Vous recevez seulement à la fin du programme le résultat, en attendant un input, ou durant une pause (**PAUSE**). Le mode rapide est 4 fois plus vite et il est employé surtout quand il a affaire à des programmes dans lesquels il y a beaucoup à calculer et peu de résultat ou quand on imprime de longs programmes.

Le changement de vitesse se fait à l'aide des touches **FAST** et **SLOW**.

Le clavier:
Le clavier ne comporte pas seulement des symboles (lettres, chiffres, etc), mais également des signes composés (mots clés, fonctions) et ces signes composés sont toujours imprimés en enfonçant une seule touche au lieu de les épeler. Pour en arriver là, certaines touches ont 5 fonctions différentes, qu'on peut obtenir en enfonçant une certaine touche et la touche **SHIFT** en même temps et en donnant à la machine des modes différents. Le mode est indiqué par le 'curseur' une lettre en vidéo inverse, qui montre où la prochaine lettre du vidéo sera placée. Mode **K** (pour mots-clés) se présente automatiquement quand l'ordinateur s'attend à une instruction ou une ligne de programme (au lieu d'éléments input), et de sa position sur la ligne il sait qu'il doit s'attendre à un numéro de ligne ou à un mot clé. Ceci se passe juste au début de la ligne, ou à quelques espaces après le début, ou juste après **THEN**. La prochaine touche, si elle n'est pas décalée (= une touche + **SHIFT**), sera interprétée comme un mot clé (écrit au-dessus de la touche) ou comme un chiffre.

Mode **L** se présente dans tous les autres cas. La prochaine touche, si pas décalée, sera interprétée comme symbole principal de cette touche.

Dans les modes **K** et **L** les touches décalées seront interprétées comme des caractères subsidiaires qui se trouvent en couleur rouge dans le coin de droite en haut de la touche.

Mode **F** (la fonction) se présente après avoir enfoncé la touche **FONCTION** (**NEWLINE** + **SHIFT**) et ne reste que le temps de frapper une touche. Cette touche sera interprétée comme nom de fonction. Les noms de fonctions se situent en-dessous des touches.

Mode **G**, se présente après avoir enfoncé la touche **GRAPHICS** (9 décalé) et cette lettre reste affichée jusqu'à ce qu'elle soit enfoncée à nouveau. Une touche non décalée donnera des caractères en vidéo inverse de son interprétation en mode **L**. Une touche décalée fera la même chose, pourvu que ce soit un symbole, mais, là où normalement une touche décalée donne un signe, dans le mode graphique elle donne un symbole graphique qui apparaît au coin de droite de la touche.

L'écran:

L'écran consiste en 24 lignes de 32 caractères et il est divisé en deux parties. La partie supérieure a 22 lignes au maximum et affiche une liste ou l'output d'un programme.

La partie inférieure, qui consiste en au-moins 2 lignes, est employée les instructions input et également pour l'affichage des rapports.

Le Input du clavier: apparaît à la partie inférieure de l'écran, juste comme c'est imprimé, chaque caractère (symbole simple ou signe composé) étant inséré juste avant le curseur. Le curseur peut être déplacé vers la gauche ⇦ ou vers la droite ⇨. Le caractère qui précède le curseur peut être effacé à l'aide de **RUBOUT** (Ø + **SHIFT**) (NOTE: toute la ligne peut être effacée avec **EDIT** (1 + **SHIFT**) et **NEWLINE**, et s'il s'agit d'éléments input, **EDIT** suffit.)

Au moment où **NEWLINE** est enfoncé, la ligne est exécutée, introduite dans le programme ou employée comme élément input, selon le cas, sauf s'il y a erreur de syntaxe. Dans ce cas, le symbole ■ apparaît juste devant l'erreur. Quand les lignes de programme sont introduites, une liste est affichée dans la partie supérieure de l'écran. La conception de la liste est assez compliquée et est décrite plus explicitement dans le chapitre 9, ex. 6. La dernière ligne introduite s'appelle la ligne courante et est indiquée par le symbole **>**. Ceci peut être modifié par les touches ⇩ et ⇧ (6 et 7 décalées). Si vous enfoncez **EDIT**, la ligne courante descend vers le bas de l'écran, et peut être "éditée".

Quand une instruction ou un programme sont exécutés, l'écran est d'abord nettoyé et ensuite le output est affiché sur la partie supérieure de l'écran et il y reste jusqu'à l'introduction du programme ou jusqu'à ce que **NEWLINE** soit imprimé suivi d'une ligne vide, ou jusqu'à ⇩ ou ⇧ ont été enfoncés. Sur la partie inférieure, un rapport de forme m/n apparaît. m est le code qui montre ce qui s'est passé (cfr. annexe B) et n'est le numéro de la dernière ligne exécutée – ou Ø pour une commande.

Le rapport reste affiché jusqu'à ce qu'une autre touche soit enfoncée (et indique le mode **K**).

Dans certains cas, la touche **SPACE** se comporte comme **BREAK**, arrêtant l'ordinateur avec le rapport D.

Ceci est visible:
(i) à la fin d'une déclaration quand le programme est exécuté,
(ii) quand l'ordinateur cherche un programme sur bande magnétique,
(iii) quand l'ordinateur emploie l'imprimante (ou quand, par accident, il essaie de l'employer quand il n'est pas branché),

Le BASIC:
Les chiffres sont emmagasinés à 9 ou 1Ø chiffres digitaux près. Le chiffre le plus grand est: $1Ø^{38}$, et le chiffre – positif – le plus petit est $4*1Ø^{-39}$. Un chiffre est emmagasiné dans le ZX81 en système binaire de la virgule flottante avec un byte exponentiel e ($1 \leq e \leq 255$) et quatre bytes mantissa m ($1/2 \leq m < 1$). Ceci représente le chiffre $m*2^{e-128}$.

Puisque $1/2 \leq m < 1$, le bit le plus significatif de la mantissa m est toujours 1. Ainsi, nous pouvons le remplacer par un bit afin de montrer le signe – Ø pour des chiffres positifs et 1 pour les chiffres négatifs. Zéro est très spécialement représenté, avec les 5 bytes égaux à Ø.

Les variables numériques ont des noms de longueur variable, commençant par une lettre et continuant par une lettre ou un chiffre. Ils ont tous une signification qui fait que **LONGNAME** et **LONGNAMETOO** sont deux noms différents. Les espaces sont ignorés.

Les variables de contrôle des boucles **FOR-NEXT** ont des noms d'une seule lettre, qui peut être identique à un nom d'une simple variable. Ils peuvent avoir n'importe quelle dimension de n'importe quelle envergure.
Les indices commencent à 1.

Les chaînes sont très flexibles en longueur. Le nom d'une chaîne se compose d'une lettre suivie de $. Les tableaux de chaînes ont n'importe quelle dimension de n'importe quelle envergure. Le nom est une seule lettre suivie d'un $ et doit être différent du nom de la chaîne. Toutes les chaînes d'un même tableau ont une longueur fixe, qui est spécifiée dans une dimension supplémentaire, finale dans la déclaration **DIM**. Les indices commencent à 1.
Trancher (= slicing): Les sous-chaînes d'une chaîne peuvent être spécifiées par un trancheur (slicer).
Un trancheur peut être:
(i) vide,
(ii) une expression numérique,
(iii) une expression numérique **TO** une expression numérique,
et il est employé pour l'expression d'une sous-chaîne à l'aide de:
a. une expression de chaîne (slicer),
b. une variable de tableau de chaîne (indice,.....indice, slicer)
ce qui veut dire la même chose que
variable de tableau de chaîne (indice,.....indice), (slicer)

Supposons que, dans a, l'expression de chaîne a la valeur s$, qui est considérée comme une sous-chaîne de soi-même.

Quand le trancheur (slicer) est une expression numérique avec la valeur m, le résultat est le même caractère de s$ (une sous-chaîne de longueur 1). Quand le trancheur a la forme de (iii), supposons que la première expression numérique a la valeur m (la valeur erronée est 1) et la seconde, la valeur n (la valeur erronée est la longueur de s$).

Si $1 \leqslant m \leqslant n$ la longueur de s$, le résultat est la sous-chaîne de s$, commençant par le m-ième caractère et terminant par le n-ième.

Si $0 \leqslant n < m$, le résultat est la chaîne vide. Sinon il y a l'erreur 3.

Le "tranchement" se fait avant l'évaluation des fonctions ou des opérations, sauf si les parenthèses le stipulent autrement.

Les sous-chaînes peuvent être affectées (cfr. **LET**).

L'argument d'une fonction n'a pas besoin de parenthèses si c'est une constante ou une variable (indicée ou tranchée).

Fonction	Type d'opérande	Résultat
	X	
ABS	Chiffre	Valeur absolue.
ACS	Chiffre	Arcocosine en radians. Erreur A si X ne se situe pas entre -1 et +1.
AND	Opération binaire Opérande de droite est tjs un chiffre Opérande numérique de gauche:	A and B = A si B ≠ Ø Ø si B = Ø
	Opérande de gauche d'une chaîne:	A$ and B = A$ si B ≠ Ø A$ si B = Ø
ASN	Chiffre	Arcsine en radians. Erreur A si X ne se situe pas entre -1 et +1.
ATN	Chiffre	Arc tangent en radians.
CHR$	Chiffre	Le caractère dont le code est X est arrondi vers le chiffre entier le plus proche. Erreur B si X ne se situe pas entre Ø et 255.
CODE	Chaîne	Le code du premier caractère dans x (ou Ø, si x est la chaîne vide).
COS	Chiffre (en radians)	Cosinus.
EXP	Chiffre	e^x.
INKEY$	Nihil	Lit le clavier. Le résultat est le caractère représenté (en mode **L**) par la touche enfoncée, sinon, c'est la chaîne vide.
INT	Chiffre	Chiffre entier (arrondi toujours vers le bas).
LEN	Chaîne	Longueur.
LN	Chiffre	Logarithme naturel (base e). Erreur A si x Ø.
NOT	Chiffre	Ø si x ≠ Ø, 1 si x = Ø.
OR	Opération binaire, les deux opérandes sont des chiffres.	A or B = 1 si B ≠ Ø A si B = Ø OR a la priorité 2.
PEEK	Chiffre	La valeur du byte en mémoire dont l'adresse est x (arrondi vers le chiffre entier le plus proche). Erreur B, si x ne se situe pas entre Ø et 65535.
PI	Nihil	π. (3,14159265...)
RND	Nihil	Le prochain chiffre pseudo-arbitraire, y, dans une séquence créée en prenant les puissances de 75 modulo 65537, soustraignant 1 et divisant par 65536. Ø y 1.
SGN	Chiffre	Signum: le signe (-1, Ø ou +1) de x.
SIN	Chiffre	Sine.

Fonction	Type d'opérande	Résultat
SQR	Chiffre	Racine carrée. Erreur B, si x Ø.
STR$	Chiffre	La chaîne de caractères qui serait affichée, si x était imprimé.
TAN	Chiffre (en radians)	Tangent.
USR	Chiffre	Appelle la sous-chaîne du code machinal, dont l'adresse de départ est x (arrondi vers le chiffre entier le plus proche). Inversément, le résultat est le contenu de la paire du registre bc. Erreur B, si x ne se situe pas entre Ø et 65535.
VAL	Chaîne	Evalue x (sans symboles de citation) comme une expression numérique. Erreur C, si x contient une erreur de syntaxe, ou s'il donne une valeur de chaîne. Les autres erreurs dépendent de l'expression.
—	Chiffre	Négation.

Les opérations suivantes sont binaires:

+	addition (pour chiffres) ou enchaînement (pour chaînes).	
-	soustraction.	
*	multiplication.	
/	division.	
**	élever à une puissance, erreur B si l'opérande de gauche est négative.	
=	égale.	
>	plus grand que.	
<	plus petit que.	Les deux opérandes doivent être du même type,
	plus petit ou égal	le résultat est un chiffre, 1, si la comparaison tient
	plus grand ou égal.	Ø, si elle ne tient pas.
	pas égal à.	

Les fonctions et les opérations ont les priorités suivantes:

Opération	**Priorité**
indice et "trancher" (slicing)	12
toutes les fonctions, excepté **NOT** et - (unaire)	11
**	10
- (unaire)	9
* , /	8
+ , - (binaire)	6
=, <, >, , , <>.	5
NOT	4
AND	3
OR	2

Fonction **Type d'opérande** **Résultat**

Déclarations

Dans cette liste,

α	représente une seule lettre.
v	représente une variable.
x, y, z,	représentent des expressions numériques.
m, n	représentent des expressions numériques qui sont arrondies vers le chiffre entier le plus proche.
e	représente une expression.

Remarquez que les expressions arbitraires sont acceptées partout (sauf comme numéro de ligne au début d'une déclaration). Toutes les déclarations, sauf **INPUT**, peuvent être employées comme instructions et aussi dans les programmes (bien qu'elles aient plus de sens dans l'une que dans l'autre).

CLEAR	Efface toutes les variables en libérant l'espace qu'elles occupaient.
CLS	(Efface l'écran). Efface tout le fichier d'affichage. Cfr. le chapitre 27 concernant le fichier d'affichage.
COPY	Envoie une copie de l'écran à l'imprimante, s'il est branché, sinon il ne fait rien. En contradiction avec les autres instructions, **COPY** n'efface pas l'écran en premier lieu. Il n'y a pas besoin d'espace avant **COPY**. Erreur D, si vous enfoncez **BREAK**.
CONT	Supposez que p/q est le dernier rapport avec un chiffre autre que zéro. Dans ce cas, **CONT** a l'effet de **GOTO** q, si $p \neq 9$.**GOTO** q+1, si $p = 9$.
DIMα (n1,....nk)	Efface tout tableau du nom de α, et crée un tableau α de chiffres avec k dimensions (n1,.......nk). Initie toutes les valeurs à Ø. Erreur 4 apparaît s'il n'y a pas d'espace pour placer le tableau. Un tableau reste indéfini aussi longtemps qu'il ne reçoit pas de dimensions par une déclaration **DIM**.
DIMα \$ (n1,...nk)	Efface tout tableau ou toute chaîne du nom α\$, et crée un tableau α \$ de chiffres avec k dimensions (n1,......nk). Initie toutes les valeurs à " ". Erreur 4 apparaît s'il n'y a pas de place pour le tableau. Un tableau reste indéfini aussi longtemps qu'il n'a pas reçu de dimensions par une déclaration **DIM**.
FAST	Commence le mode rapide dans lequel le fichier d'affichage n'apparaît qu'à la fin du programme, au moment où les éléments **INPUT** sont introduits, ou durant une pause.

Fonction	Type d'opérande	Résultat
FORα = X **TO** Y		**FOR**α = X **TO** Y **STEP** 1.
FORα = X **TO** Y **STEP** Z		Efface n'importe quelle variable simple α et crée une variable de contrôle de la valeur de X, limite Y, phase Z et une adresse de boucle qui dépasse le numéro de ligne de la déclaration **FOR** par 1 (-1 s'il s'agit d'une commande). Contrôlez si la valeur initiale est plus grande (si la phase Ø) ou moins grande (si la phase < Ø) que la limite, et si cela est le cas, sautez à la déclaration **NEXT** au début d'une ligne. L'erreur 4 apparaît s'il n'y a pas de place pour la variable de contrôle.
GOSUBn		Pousse le numéro de ligne de la déclaration **GOSUB** sur le 'tas'. Ensuite, comme **GOTO**n. L'erreur 4 apparaîtra, s'il n'y a pas assez de **RETURNS**s.
GOTOn		Saute à la ligne n (ou, s'il n'y en a pas, vers la ligne suivante).
IF x **THEN** s		Si x est vrai, (pas zéro), s'est exécuté.
INPUTv		Arrête tout et attend jusqu'à ce que l'usager imprime une expression. Sa valeur est affectée à v. En mode rapide, le fichier d'affichage est donné. **INPUT** ne peut pas être utilisé comme instruction. Si vous essayez, vous recevez l'erreur 8. Si le premier caractère de la ligne **INPUT** est **STOP**, le programme s'arrête avec l'erreur D.
LET v = e		Affecte la valeur de e à la variable v. **LET** ne peut pas être omis. Une variable simple reste indéfinie jusqu'à ce qu'elle ait été affectée à une déclaration **LET** ou **INPUT**. Si v est une variable indicée de chaîne, ou une variable de chaîne tranchée (sliced) nous avons affaire à une déclaration à la façon de Procrust. La valeur de la chaîne de e est ou bien tronquée ou bien remplie d'espaces à la droite, pour faire la même longueur que la variable v.
LIST		**LIST Ø**
LIST n		Affiche le programme au téléviseur, commençant à la ligne n et fait de n la ligne courante. L'erreur 4 ou 5 apparaîssent si le fichier est trop long pour apparaître à l'écran. **CONT** refera exactement la même chose.
LLIST		**LLIST Ø**.
LLIST n		Comme **LIST**, mais en employant l'imprimante au lieu du téléviseur. Ne faites rien, si l'imprimante n'est pas branchée. S'arrête avec le rapport D, si on frappe **BREAK**.
LOAD f		Cherche un programme f sur bande magnétique, charge ce programme et ses variables. Si f = " ", vous chargez le premier programme disponible. Si **BREAK** est enfoncé, ou si une erreur est découverte sur la bande:

Fonction	Type d'opérande	Résultat
		(i) si aucun programme a été lu, la bande s'arrête avec le rapport D et l'ancien programme est affiché. (ii) si une partie du programme a été lu, ça exécute **NEW**.
LPRINT		Comme **PRINT**, mais avec l'imprimante au lieu de la télévision. Une ligne de texte est envoyée à l'imprimante: (i) quand, en imprimant, on passe d'une ligne à une autre, (ii) après une déclaration **LPRINT** qui ne se termine pas en virgule ou en point-virgule, (iii) quand une virgule ou un élément **TAB** requiert une nouvelle ligne, ou (iv) à la fin d'un programme, s'il reste encore quelque chose à imprimer. Dans un élément **AT**, seul un numéro de colonne a un effet, le numéro de ligne est ignoré (excepté que les mêmes conditions d'erreur apparaîssent que pour **PRINT**, s'il est hors de portée). Un élément **AT** n'envoie jamais une ligne de texte à l'imprimante. Il n'y aura pas d'effet quand il n'y a pas d'imprimante. L'erreur 3 apparaît, si **BREAK** est enfoncé.
NEW		Fait recommencer le système BASIC, détruisant les programmes et les variables et en employant la mémoire jusqu'au point qui n'inclut pas les bytes dont l'adresse se trouve dans la variable de système RAMTOP (bytes 16388 et 16389).
NEXTα		Trouve la variable de contrôle . Ajoute la phase à la valeur. Si la phase ⩾ Ø et la valeur > la limite ou, si la phase < Ø et la valeur < la limite, alors **NEXT** saute vers la ligne du boucle. L'erreur 1 apparaît, s'il y a une variable simple α, l'erreur 2 s'il n'y a pas une variable simple α, ni une variable de contrôle .
PAUSE n		Arrête l'ordinateur et montre le fichier d'affichage à n images (à 50 images par seconde) ou jusqu'à ce qu'une touche soit enfoncée. Ø⩽ n ⩽ 65535, sinon, l'erreur B apparaît. Si n 32767, la pause n'est pas minutée, mais continue jusqu'à ce qu'une touche soit enfoncée.
PLOT m,n		Noircit le 'pixel' (m , n); et déplace la position **PRINT** jusqu'après le 'pixel'. Ø|m|⩽63,Ø|n| ⩽ 43, sinon l'erreur B apparaît.

Fonction	Type d'opérande	Résultat
POKE m,n		Ecrit la valeur n au byte emmagasiné à l'adresse m. $\emptyset \leqslant m \leqslant 65535$, $-255 \leqslant$ n 255, sinon, l'erreur B apparaît.
PRINT...		Le "..." est une séquence d'éléments **PRINT**, séparés par des virgules ou des points-virgules, et enregistrés aux fichier d'affichage pour être montrés au téléviseur.

La position (ligne et colonne) où la prochaine ligne doit être imprimée s'appelle la position **PRINT**.
Un élément **PRINT** peut être:
(i) vide, c.-à-d. rien.
(ii) une expression numérique.
Un signe minuscule est imprimé si la valeur est négative.
Supposons que X est le modulus de la valeur.
Si x $\leqslant 1\emptyset^{-5}$, ou X $\geqslant 1\emptyset^{13}$, il est imprimé en utilisant des notations scientifiques. La partie mantissa peut s'élever à 8 chiffres digitaux (sans zéros) et le point décimal se trouve après le premier chiffre. Le point décimal est absent quand il n'y a qu'un chiffre digital. La partie exponentielle est E, suivie de + ou de –, suivie d'un ou deux chiffres digitaux. Sinon, X est imprimé de façon ordinaire en système décimal, s'élevant à 8 chiffres digitaux significatifs, et sans qu'un zéro suit le point décimal. Un point décimal au début d'un chiffre est toujours suivi d'un zéro, par ex.: .Ø3 et Ø.3 sont imprimés comme tels. Ø est imprimé comme un simple chiffre digital Ø.
(iii) Une expression de chaîne. Le nombre de signes dans la chaîne est augmentée, éventuellement avec un espace au début de la chaîne et un espace à la fin.
Les symboles de citation sont imprimés comme : ".
Les caractères non-utilisés et les caractères de contrôle comme : ?
(iv) **AT** m,n .
La position **PRINT** a changé de place vers la ligne |m| (à partir du haut), colonne n à partir de la gauche).
$\emptyset \leqslant m \leqslant 21$, sinon l'erreur 5 apparaît.
m = 22 ou 23, sinon, l'erreur B apparaît.
Ø 11 31, sinon, l'erreur B apparaît.
(v) **TAB** n
n est un modulo 32 réducé. Ensuite la position **PRINT** est envoyée à la n-ième colonne, tout en restant à la même ligne. Si cela implique un recul, elle avance vers la ligne suivante.
$\emptyset \leqslant n \leqslant 255$, sinon l'erreur B apparaît.
Un point-virgule entre deux éléments laisse la position

Fonction	Type d'opérande	Résultat
		PRINT inchangée, donc le second élément suit immédiatement le premier. Une virgule déplace la position **PRINT** d'au moins une place et ensuite encore autant de places qu'il soit nécessaire pour qu'elle reste dans la colonne Ø ou 16, même s'il est nécessaire de commencer une nouvelle ligne. Une nouvelle ligne est commencée à la fin d'une déclaration, si elle ne finit pas par un point ou un point-virgule. L'erreur 4 peut apparaître quand on ne dispose pas d'une mémoire assez grande (3K ou moins). L'erreur 5 apparaît quand l'écran et plein. Dans les deux cas, le remède est **CONT**, qui balaie l'écran et continue ensuite.
RAND		**RAND Ø**
RAND n		Mets la prochaine variable de système (**SEED**), qui est employé pour générer la prochaine valeur de **RND**. Si n ≠ Ø, **SEED** donne la valeur n ; si n = Ø, on lui donne la valeur d'une autre variable de système (**FRAME** = imafe), qui compte les images affichées à l'écran jusqu'à ce moment. L'erreur B apparaît si n ne se situe pas entre Ø et 65535.
REM...		N'a aucun effet. '...' peut être n'importe quelle séquence de caractères, excepté **NEWLINE**.
RETURN		Sort un numéro de ligne du 'tas' **GOSUB** et saute à la ligne suivante. L'erreur 7 apparaît s'il n'y a pas de numéro de ligne dans le 'tas'. Il y a une erreur dans votre programme, **GOSUBs** ne sont pas tout à fait en équilibre avec **RETURNs**.
RUN		**RUN Ø**
RUN n		**CLEAR** et ensuite **GOTO** n.
SAVE f		Enregistre les programmes et les variables sur bandes magnétiques. **SAVE** ne doit pas être employé dans un programme **GOSUB**. L'erreur F apparaît, si f est la chaîne vide, ce qui n'est pas admis.
SCROLL		Déplace le fichier d'affichage d'une ligne, perdant la ligne du haut et créant une ligne vide en bas de l'écran. P.S. La dernière ligne est réellement vide, il n'y a qu'un seul caractère (**NEWLINE**) sans espaces. (Cfr. chap. 27).
SLOW		Mets l'ordinateur dans le mode de 'calcul et affichage'. Ainsi le fichier d'affichage est continuellement montré. Le calcul est fait pendant les espaces qui apparaissent en haut et en bas de l'écran.
STOP		Arrête le programme avec le rapport 9. **CONT** résume avec la prochaine ligne.
UNPLOT		Comme **PLOT**. Blanchit le 'pixel' au lieu de le noircir.

Index

Cet index reprend les touches du clavier et la façon de s'en servir — K, L, F ou G en position majuscules ou non) ainsi que leurs codes.

Habituellement, une indication n'est citée qu'une fois par chapitre ; par conséquent, lorsque vous avez trouvé une référence, vous devez lire le reste du chapitre, exercices compris.

C

Les accessoires suivants sont disponibles :

Mémoire 16 K
Mémoire 64 K
Printer
Clavier
Couvercle clavier
Manuel français
Nederlandse handleiding
Module 1/0 + 5 K
Rouleau papier printer
Cassette : n° 1 jeux
n° 2 formation
n° 3 enterprise
n° 4 jeux
n° 5 formation
n° 6 échec
n° 7 VU clac
n° 8 OTHELLO
n° 9 pedagogix 81
n° 10 backgammon
n° 11 space raiders
n° 12 Vu file
n° 13 The collector's Pack
n° 14 Biorhytms
KIT ZX 81

Imprimerie S.E.I.P. - 36, rue Thiéfry - 1030 Bruxelles - Tél. 216.75.75

sinclair

ZX81

PERSONAL COMPUTER

DRION

COMPUTERS

AMSTRAD CPC

OCR

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/